

6.1 User-defined function basics

Functions (general)

A program may perform the same operation repeatedly, causing a large and confusing program due to redundancy. Program redundancy can be reduced by creating a grouping of predefined statements for repeatedly used operations, known as a **function**. Even without redundancy, functions can prevent a main program from becoming large and confusing.

PARTICIPATION ACTIVITY

6.1.1: Functions can reduce redundancy and keep the main program simple.



Animation content:

undefined

Animation captions:

1. Commonly, a program performs the same operation, such as a calculation, in multiple places. Here, the Fahrenheit to Celsius calculation is done in three places.
2. Repeated operations clutter the main program. And such repeated operations are more prone to errors.
3. A better approach defines the Fahrenheit to Celsius calculation once, named F2C here. Then, F2C can be "called" three times, yielding a simpler main program.
4. The impact is even greater when the operation has multiple statements -- here 3 statements, but commonly tens of statements. The main program is much simpler.
5. Even without repeated operations, calling predefined operations keeps the main program simple and intuitive.

PARTICIPATION ACTIVITY

6.1.2: Reasons for functions.



Consider the animation above.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 1) In the original main program, the Fahrenheit to Celsius calculation appeared how many times?

- 1
- 3



2)



Along with yielding a simpler main program, using the predefined Fahrenheit to Celsius calculation prevented what error in the original program?

- Adding rather than subtracting 32.0
 - Multiplying by 9.0 / 5.0 rather than by 5.0 / 9.0
- 3) In the last example above, the main program was simplified by ____.
- eliminating redundant code for operation XYZ
 - predefining operations for XYZ and PQR

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



Basics of functions

A **function** is a named list of statements.

- A **function definition** consists of the new function's name and a block of statements. Ex:
`double CalcPizzaArea() { /* block of statements */ }`
- A **function call** is an invocation of a function's name, causing the function's statements to execute.

The function's name can be any valid identifier. A **block** is a list of statements surrounded by braces.

Below, the function call `CalcPizzaArea()` causes execution to jump to the function's statements. Execution returns to the original location after executing the function's last statement.

PARTICIPATION ACTIVITY

6.1.3: Function example: Calculating pizza area.



Animation content:

undefined

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Animation captions:

1. The function call `CalcPizzaArea()` jumps execution to the function's statements.
2. After the last statement of the `CalcPizzaArea()` function, execution returns to the original location and the area of the pizza is returned and printed in `main()`.

**PARTICIPATION
ACTIVITY**

6.1.4: Function basics.



Given the `CalcPizzaArea()` function defined above and the following `main()` function:

```
int main(void) {
    printf("12 inch pizza is %lf square inches\n", CalcPizzaArea());
    printf("12 inch pizza is %lf square inches\n", CalcPizzaArea());
    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 1) How many function calls to `CalcPizzaArea()` exist in `main()`?

Check**Show answer**

- 2) How many function definitions of `CalcPizzaArea()` exist *within* `main()`?

Check**Show answer**

- 3) How many output statements exist in `main()`?

Check**Show answer**

- 4) How many output statements exist in `CalcPizzaArea()`?

Check**Show answer**

- 5) Is `main()` itself a function definition? Answer yes or no.

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



Returning a value from a function

A function may return one value using a **return statement**. Below, the ComputeSquare() function is defined to have a return type of int; thus, the function's return statement must have an expression that evaluates to an int.

PARTICIPATION ACTIVITY

6.1.5: Function returns computed square.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021



Animation captions:

1. ComputeSquare() is called, passing 7 to the function's numToSquare parameter.
2. The function computes the square of the parameter numToSquare.
3. ComputeSquare(7) evaluates to 49, which is then assigned to numSquared.

Other return types are allowed, such as char, double, etc. A function can only return one item, not two or more. A return type of **void** indicates that a function does not return any value.

PARTICIPATION ACTIVITY

6.1.6: Return.



Given the definition below, indicate which are valid return statements:

```
int CalculateSomeValue(int num1, int num2) { ... }
```

1) `return 9;`

- Yes
 No

2) `return num1;`

- Yes
 No

3) `return (num1 + num2) + 1;`

- Yes
 No

4) `return;`

- Yes
 No

5) `return num1 num2;`©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Yes No6) `return (0);` Yes No

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

PARTICIPATION ACTIVITY

6.1.7: More on return.



1) The following is a valid function definition:

`char GetItem() { ... }` True False

2) The following is a valid function definition for a function that returns two items:

`int, int GetItems() { ... }` True False

3) The following is a valid function definition:

`void PrintItem() { ... }` True False

Parameters

A programmer can influence a function's behavior via an input.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

- A **parameter** is a function input specified in a function definition. Ex: A pizza area function might have diameter as an input.
- An **argument** is a value provided to a function's parameter during a function call. Ex: A pizza area function might be called as `CalcPizzaArea(12.0)` or as `CalcPizzaArea(16.0)`.

A parameter is like a variable declaration. Upon a call, the parameter's memory location is allocated, and the parameter is assigned with the argument's value. Upon returning to the original call location, the parameter is deleted from memory.

An argument may be an expression, like 12.0, x, or x * 1.5.

PARTICIPATION ACTIVITY

6.1.8: Function with parameters example: Calculating pizza area for different diameters.

**Animation content:**

undefined

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Animation captions:

1. The function call CalcPizzaArea() jumps execution to the function's statements, passing 12.0 to the function's pizzaDiameter parameter.
2. The next function call passes 16.0 to the function's pizzaDiameter parameter, which results in a different pizza area.

PARTICIPATION ACTIVITY

6.1.9: Parameters.



- 1) Complete the function beginning to have a parameter named numServings of type int.

```
int CalcCalories(  
    ) {
```

Check**Show answer**

- 2) Write a statement that calls a function named CalcCalories, passing the value 3 as an argument.

Check**Show answer**

- 3) Is the following a valid function definition beginning? Type yes or no.

```
int MyFct(int userNum + 5)  
{ ... }
```

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021





4) Assume a function int

```
GetBirthdayAge(int userAge)
simply returns the value of userAge
+ 1. What will the following output?
```

```
printf("%d",
GetBirthdayAge(42));
printf("%d",
GetBirthdayAge(20));
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

[Show answer](#)

Multiple or no parameters

A function definition may have multiple parameters, separated by commas. Parameters are assigned with argument values by position: First parameter with first argument, second with second, etc.

A function definition with no parameters must still have the parentheses, as in:

`int DoSomething() { ... }`. Good practice is to use the keyword void for an empty parameter list, as in: `int DoSomething(void) { ... }`. The call must include parentheses, with no argument, as in: `DoSomething()`.

Figure 6.1.1: Function with multiple parameters.

```
#include <stdio.h>

double CalcPizzaVolume(double pizzaDiameter, double pizzaHeight) {
    double pizzaRadius;
    double pizzaArea;
    double pizzaVolume;
    double piVal = 3.14159265;

    pizzaRadius = pizzaDiameter / 2.0;
    pizzaArea = piVal * pizzaRadius * pizzaRadius;
    pizzaVolume = pizzaArea * pizzaHeight;
    return pizzaVolume;
}

int main(void) {
    printf("12.0 x 0.3 inch pizza is %lf inches cubed.\n", CalcPizzaVolume(12.0, 0.3));
    printf("12.0 x 0.8 inch pizza is %lf inches cubed.\n", CalcPizzaVolume(12.0, 0.8));
    printf("16.0 x 0.8 inch pizza is %lf inches cubed.\n", CalcPizzaVolume(16.0, 0.8));
    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
12.0 x 0.3 inch pizza is 33.929201 inches cubed.
12.0 x 0.8 inch pizza is 90.477868 inches cubed.
16.0 x 0.8 inch pizza is 160.849544 inches cubed.
```

PARTICIPATION ACTIVITY

6.1.10: Multiple parameters.



- 1) Which correctly defines two integer parameters x and y for a function definition:

`int CalcVal(...)?`

- (int x; int y)
- (int x, y)
- (int x, int y)

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 2) Which correctly passes two integer arguments for the function call:

`CalcVal(...)?`

- (99, 44 + 5)
- (int 99, 44)
- (int 99, int 44)



- 3) Given a function definition:

`int CalcVal(int a, int b, int c)`

b is assigned with what value during this function call:

`CalcVal(42, 55, 77);`

- Unknown
- 42
- 55



- 4) Given a function definition:

`int CalcVal(int a, int b, int c)`

and given int variables i, j, and k, which are valid arguments in the call

`CalcVal(...)?`

- (i, j)
- (k, i + j, 99)
- (i + j + k)

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION

ACTIVITY

6.1.11: Calls with multiple parameters.



Given:

```
int GetSum(int num1, int num2) {  
    return num1 + num2;  
}
```

- 1) What will be returned for the following function call?

GetSum(1, 2);

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

- 2) Write a statement that calls GetSum() to return the sum of x and 400 (providing the arguments in that order). End with ;

Check**Show answer**

Calling functions from functions

A function's statements may call other functions. In the example below, the PizzaCalories() function calls the CalcCircleArea() function. (Note that main() itself is the first function called when a program executes, and calls other functions.)

Figure 6.1.2: Functions calling functions.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
#include <stdio.h>

double CalcCircleArea(double circleDiameter) {
    double circleRadius;
    double circleArea;
    double piVal = 3.14159265;

    circleRadius = circleDiameter / 2.0;
    circleArea = piVal * circleRadius * circleRadius;

    return circleArea;
}

double PizzaCalories(double pizzaDiameter) {
    double totalCalories;
    double caloriesPerSquareInch = 16.7;      // Regular crust pepperoni pizza

    totalCalories = CalcCircleArea(pizzaDiameter) * caloriesPerSquareInch;

    return totalCalories;
}

int main() {
    printf("12 inch pizza has %.2lf calories.\n", PizzaCalories(12.0));
    printf("14 inch pizza has %.2lf calories.\n", PizzaCalories(14.0));

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

12 inch pizza has 1888.73 calories.
14 inch pizza has 2570.77 calories.

PARTICIPATION ACTIVITY

6.1.12: Functions calling functions.



Complete the PizzaCaloriesPerSlice() function to compute the calories for a single slice of pizza. A PizzaCalories() function returns a pizza's total calories given the pizza diameter passed as an argument. A PizzaSlices() function returns the number of slices in a pizza given the pizza diameter passed as an argument.

```
double PizzaCaloriesPerSlice(double pizzaDiameter) {
    double totalCalories;
    double caloriesPerSlice;

    totalCalories = Placeholder_A;
    caloriesPerSlice = Placeholder_B;

    return caloriesPerSlice;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 1) Type the expression for Placeholder_A to compute the total



calories for a pizza with diameter
pizzaDiameter.

```
totalCalories =  
    ;
```

Check**Show answer**

- 2) Type the expression for Placeholder_B to compute the calories per slice.

```
caloriesPerSlice =  
    ;
```

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Exploring further:

- [Functions tutorial](#) from cprogramming.com

CHALLENGE ACTIVITY

6.1.1: Basic function call.



Complete the function definition to return the hours given minutes. Output for sample program:

3.500000

347282.2052980.qx3zqy7

```
1 #include <stdio.h>  
2  
3 double GetMinutesAsHours(double origMinutes) {  
4     /* Your solution goes here */  
5 }  
6  
7 int main(void) {  
8     double minutes;  
9     scanf("%lf", &minutes);  
10    // Will be run with 210.0, 3600.0, and 0.0.  
11    printf("%f\n", GetMinutesAsHours(minutes));  
12}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```

16
17     return 0;
18 }
```

Run**CHALLENGE ACTIVITY**

6.1.2: Enter the output of the returned value.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

347282.2052980.qx3zqy7

Start

Type the program's output

```

#include <stdio.h>
int ChangeValue(int x) {
    return x + 2;
}

int main(void) {
    printf("%d\n", ChangeValue(3));

    return 0;
}
```

5

1

2

Check**Next****CHALLENGE ACTIVITY**

6.1.3: Function definition: Volume of a pyramid.



Define a function `PyramidVolume` with double data type parameters `baseLength`, `baseWidth`, and `pyramidHeight`, that returns as a double the volume of a pyramid with a rectangular base. Relevant geometry equations:

$\text{Volume} = \text{base area} \times \text{height} \times 1/3$

$\text{Base area} = \text{base length} \times \text{base width}$.

(Watch out for integer division).

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

347282.2052980.qx3zqy7

```

1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
```

```
5     double userVolume; 
6     double userLength;
7     double userWidth;
8     double userHeight;
9
10    scanf("%lf", &userLength);
11    scanf("%lf", &userWidth);
12    scanf("%lf", &userHeight);
13
14    printf("Volume: %f\n", PyramidVolume(userLength, userWidth, userHeight) );
15
16    return 0;
17 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Run

6.2 Print functions

Printing from a function

A common operation for a function is to print text. Large text outputs can clutter the main() function of a program, especially if the text needs to be output multiple times. A function that only prints typically does not return a value. The **void** keyword indicates a function does not return a value. A function with a void return type is often called a **void function**. Once a void function finishes execution, control returns back to the caller and no value is returned.

PARTICIPATION
ACTIVITY

6.2.1: Printing with a void function.



Animation content:

undefined

Animation captions:

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

1. Printing instructions can clutter a program.
2. A print function can handle output and reduce clutter in main().
3. main() calls function PrintSummary(), which prints the parameters as formatted output.
4. PrintSummary() completes execution and returns back to the caller, main().

PARTICIPATION

6.2.2: Print functions.



ACTIVITY

1) Print operations must be performed in main().

- True
- False

2) A void function can have any number of parameters.

- True
- False

3) A print function must return the value that was output.

- True
- False

4) A void function can be defined with either "void" or no type specifier.

- True
- False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

A function that produces output can also return a value, but this material separates these operations for clarity. A function that both outputs and returns a value must have a return type that is not void.

Calling a print function multiple times

One benefit of a print function is that complex output statements can be written in code once. Then the print function can be called multiple times to produce the output instead of rewriting complex statements for every necessary instance. Changes to output and formatting are made easier and are less prone to error.

PARTICIPATION ACTIVITY

6.2.3: Calling a print function repeatedly.

Animation content:

Step 1: Two programs shown, both output the phrase, "The Greatest BLANK on Earth". Program A has a main() function that outputs the phrase with a string variable in place of the blank. Program B has a main() that calls PrintGreatest(), a function with a string parameter. PrintGreatest() outputs the phrase with the argument in place of the blank. Step 2: Consider printing the phrase again with a different string in place of the blank. A programmer editing Program A must duplicate the entire phrase in main(). A programmer editing Program B just needs to call PrintGreatest() with the new string. Step 3: Consider changing the formatting of the phrase, e.g. adding '!' to the end. A programmer editing Program B makes a single change in the PrintGreatest() function. Step 4: A programmer editing Program A must change every instance of the phrase. Here there are two.

Animation captions:

1. A print function can improve the organization of a program.
2. The benefit of a print function increases with repeated calls. Both Program A and B output the formatted text twice, but the output code for Program B is only written once in the PrintGreatest() function.
3. In Program B, any changes to the output, e.g. adding '!', only have to be made in the PrintGreatest() function.
4. Without a function, like in Program A, output must be changed in multiple instances, which can be time-consuming and lead to errors.

PARTICIPATION ACTIVITY

6.2.4: Calling a print function multiple times.



Refer to the example programs above.

- 1) To output the phrase, "The Greatest [word] on Earth", with ten different words, what is the minimum number of times main() would need to call PrintGreatest() in Program B?



- 1
- 10
- 30

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 2) To output "in the Galaxy" instead of "on Earth" in the phrase (Ex: "--The Greatest *Cafe* in the Galaxy!-- "), how many statements need to be changed in Program A and Program B?



- Program A: 1
Program B: 1
- Program A: 2
Program B: 1
- Program A: 2
Program B: 2

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Example: Menu system

Figure 6.2.1: Example: Menu System.

```
#include <stdio.h>
#include <stdbool.h>

void PrintMenu() {
    printf("Today's Menu:\n");
    printf("  1) Gumbo\n");
    printf("  2) Jambalaya\n");
    printf("  3) Quit\n\n");
}

int main() {
    bool quit = false;
    int choice;

    while (!quit) {
        PrintMenu();
        printf("Enter choice: ");
        scanf("%d", &choice);
        printf("\n");
        if (choice == 3) {
            printf("Goodbye\n");
            quit = true;
        }
        else {
            printf("Order: ");
            if (choice == 1) {
                printf("Gumbo\n");
            }
            else if (choice == 2) {
                printf("Jambalaya\n");
            }
            printf("\n");
        }
    }
    return 0;
}
```

```
Today's Menu:
  1) Gumbo
  2) Jambalaya
  3) Quit

Enter choice: 2
Order: Jambalaya

Today's Menu:
  1) Gumbo
  2) Jambalaya
  3) Quit

Enter choice: 1
Order: Gumbo

Today's Menu:
  1) Gumbo
  2) Jambalaya
  3) Quit

Enter choice: 3
Goodbye
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

PARTICIPATION ACTIVITY

6.2.5: Example: Menu System.



Consider the example above.



1) How many times is PrintMenu() called?

- 1
- 2
- 3

2) Which of the following code statements if added to PrintMenu() would produce a compiler error?

- int numOptions;
- return 0;
- return;

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Exploring further:

- [Functions tutorial](#) from cprogramming.com

CHALLENGE ACTIVITY

6.2.1: Function parameters.



347282.2052980.qx3zqy7

Start

Type the program's output

```
#include <stdio.h>

void PrintAge(int userAge) {
    printf("You are %d\n", userAge);
}

int main(void) {
    int ageToPrint;

    ageToPrint = 20;
    PrintAge(ageToPrint);

    return 0;
}
```

You are 20

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

1

2

3

4

Check

Next

CHALLENGE ACTIVITY**6.2.2: Function call with parameter: Printing formatted measurement.**

Define a function PrintFeetInchShort(), with int parameters numFeet and numInches, that prints using ' and " shorthand. End with a newline. Remember that "\n" in a string literal starts a new line. Ex: PrintFeetInchShort(5, 8) prints:

5' 8"

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Hint: Use \" to print a double quote.

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
6     int userFeet;
7     int userInches;
8
9     scanf("%d", &userFeet);
10    scanf("%d", &userInches);
11
12    PrintFeetInchShort(userFeet, userInches); // Will be run with (5, 8), then (4,
13
14    return 0;
15 }
```

Run

6.3 Reasons for defining functions

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Improving program readability

Programs can become hard for humans to read and understand. Decomposing a program into functions can greatly aid program readability, helping yield an initially correct program, and easing future maintenance. Below, the program with functions has a main() that is easier to read and understand. For larger programs, the effect is even greater.

Figure 6.3.1: User-defined functions make main() easy to understand.

```
#include <stdio.h>

double StepsToMiles(int numSteps) {
    const double FEET_PER_STEP = 2.5; // Typical adult
    const int    FEET_PER_MILE = 5280;

    return numSteps * FEET_PER_STEP * (1.0 / FEET_PER_MILE);
}

double StepsToCalories(int numSteps) {
    const double STEPS_PER_MINUTE = 70.0; // Typical adult
    const double CALORIES_PER_MINUTE_WALKING = 3.5; // Typical adult
    double minutesTotal;
    double caloriesTotal;

    minutesTotal = numSteps / STEPS_PER_MINUTE;
    caloriesTotal = minutesTotal * CALORIES_PER_MINUTE_WALKING;

    return caloriesTotal;
}

int main(void) {
    int stepsWalked;

    printf("Enter number of steps walked: ");
    scanf("%d", &stepsWalked);

    printf("Miles walked: %lf\n", StepsToMiles(stepsWalked));
    printf("Calories: %lf\n", StepsToCalories(stepsWalked));

    return 0;
}
```

```
Enter number of steps walked: 1600
Miles walked: 0.757576
Calories: 80.000000
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Figure 6.3.2: Without program functions: main() is harder to read and understand.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
#include <stdio.h>

int main(void) {
    int stepsWalked;
    const double FEET_PER_STEP = 2.5; // Typical adult
    const int FEET_PER_MILE = 5280;
    const double STEPS_PER_MINUTE = 70.0; // Typical adult
    const double CALORIES_PER_MINUTE_WALKING = 3.5; // Typical adult
    double minutesTotal;
    double caloriesTotal;
    double milesWalked;

    printf("Enter number of steps walked: ");
    scanf("%d", &stepsWalked);

    milesWalked = stepsWalked * FEET_PER_STEP * (1.0 / FEET_PER_MILE);
    printf("Miles walked: %lf\n", milesWalked);

    minutesTotal = stepsWalked / STEPS_PER_MINUTE;
    caloriesTotal = minutesTotal * CALORIES_PER_MINUTE_WALKING;
    printf("Calories: %lf\n", caloriesTotal);

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021**PARTICIPATION ACTIVITY**

6.3.1: Improved readability.



Consider the above examples.

- 1) In the example *without* functions, how many statements are in main()?

- 6
- 16



- 2) In the example *with* functions, how many statements are in main()?

- 6
- 16



- 3) Which has fewer *total* lines of code (including blank lines), the program with or without functions?

- With
- Without

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

 Same

- 4) The program with functions called the functions directly in output statements.
Did the program without functions directly put calculations in output statements?

- No
 Yes

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Modular and incremental program development

Programmers commonly use functions to write programs modularly and incrementally.

- **Modular development** is the process of dividing a program into separate modules that can be developed and tested separately and then integrated into a single program.
- **Incremental development** is a process in which a programmer writes, compiles, and tests a small amount of code, then writes, compiles, and tests a small amount more (an incremental amount), and so on.
- A **function stub** is a function definition whose statements have not yet been written.

A programmer can use function stubs to capture the high-level behavior of main() and the required function (or modules) before diving into details of each function, like planning a route for a road trip before starting to drive. A programmer can then incrementally develop and test each function independently.

PARTICIPATION ACTIVITY

6.3.2: Function stub used in incremental program development.



Animation captions:

1. main() captures the program's high-level behavior and makes calls to three functions stubs.
The program can be compiled and tested before writing and testing those functions.
2. The programmer then writes and tests the ConvKilometersToMiles() and ConvLitersToGallons() functions.
3. After testing the ConvKilometersToMiles() and ConvLitersToGallons() functions, the programmer can then complete the program by writing and testing the CalcMpg() function.

PARTICIPATION ACTIVITY

6.3.3: Incremental development.



- 1) Incremental development may involve more frequent compilation, but



ultimately lead to faster development of a program.

- True
 - False
- 2) The program above does not compile because CalcMpg() is a function stub.
- True
 - False
- 3) A key benefit of function stubs is faster running programs.
- True
 - False
- 4) Modular development means to divide a program into separate modules that can be developed and tested independently and then integrated into a single program.
- True
 - False



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



zyDE 6.3.1: Function stubs.

Complete the program by writing and testing the CalcMpg() function.

Load default template

```
1 #include <stdio.h>
2 // Program converts a trip's kilometers and Liters into miles, gallons, and
3
4 double ConvKilometersToMiles(double numKm) {
5     double milesPerKm = 0.621371;
6     return numKm * milesPerKm;
7 }
8
9 double ConvLitersToGallons(double numLiters) {
10    double gallonsPerLiter = 0.264172;
11    return numLiters * gallonsPerLiter;
12 }
13
14 double CalcMpg(double distMiles, double gasGallons) {
15     printf("FIXME: Calculate MPG\n");
16     return 0.0;
17 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

18

30.6 10 2 0

Run

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Avoid writing redundant code

A function can be defined once, then called from multiple places in a program, thus avoiding redundant code. Examples of such functions are math functions like `abs()` that relieve a programmer from having to write several lines of code each time an absolute value needs to be computed.

The skill of decomposing a program's behavior into a good set of functions is a fundamental part of programming that helps characterize a good programmer. Each function should have easily-recognizable behavior, and the behavior of `main()` (and any function that calls other functions) should be easily understandable via the sequence of function calls.

A general guideline (especially for beginner programmers) is that a function's definition usually shouldn't have more than about 30 lines of code, although this guideline is not a strict rule.

PARTICIPATION ACTIVITY

6.3.4: Redundant code can be replaced by multiple calls to one function.



Animation content:

undefined

Animation captions:

1. Circle area is calculated twice, leading to redundant code.
2. The redundant code can be replaced by defining a `CalcCircleArea` function.
3. Then `main` is simplified by calling the `CalcCircleArea()` function from multiple places in the program.

©zyBooks 12/22/21 12:35 1026490

UACS100Fall2021

PARTICIPATION ACTIVITY

6.3.5: Reasons for defining functions.



- 1) A key reason for creating functions is



to help main() run faster.

- True
 - False
- 2) Avoiding redundancy means to avoid calling a function from multiple places in a program. □
- True
 - False
- 3) If a function's internal statements are revised, all function calls will have to be modified too. □
- True
 - False
- 4) A benefit of functions is to increase redundant code. □
- True
 - False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

CHALLENGE ACTIVITY

6.3.1: Functions: Factoring out a unit-conversion calculation. □

Write a function so that the main() code below can be replaced by the simpler code that calls function MphAndMinutesToMiles(). Original main():

```
int main(void) {
    double milesPerHour;
    double minutesTraveled;
    double hoursTraveled;
    double milesTraveled;

    scanf("%lf", &milesPerHour);
    scanf("%lf", &minutesTraveled);

    hoursTraveled = minutesTraveled / 60.0;
    milesTraveled = hoursTraveled * milesPerHour;

    printf("Miles: %lf\n", milesTraveled);

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
```

```
3 /* Your solution goes here */
4
5 int main(void) {
6     double milesPerHour;
7     double minutesTraveled;
8
9     scanf("%lf", &milesPerHour);
10    scanf("%lf", &minutesTraveled);
11
12    printf("Miles: %lf\n", MphAndMinutesToMiles(milesPerHour, minutesTraveled));
13
14    return 0;
15 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Run**CHALLENGE ACTIVITY**

6.3.2: Function stubs: Statistics.



Define stubs for the functions called by the below main(). Each stub should print "FIXME: Finish FunctionName()" followed by a newline, and should return -1. Example output:

```
FIXME: Finish GetUserNum()
FIXME: Finish GetUserNum()
FIXME: Finish ComputeAvg()
Avg: -1
```

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
6     int userNum1;
7     int userNum2;
8     int avgResult;
9
10    userNum1 = GetUserNum();
11    userNum2 = GetUserNum();
12
13    avgResult = ComputeAvg(userNum1, userNum2);
14
15    printf("Avg: %d\n", avgResult);
16
17    return 0;
18 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Run

6.4 Writing mathematical functions

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Mathematical functions

A function is commonly defined to compute a mathematical calculation involving several numerical parameters and returning a numerical result. The program below uses a function to convert a person's height in U.S. units (feet and inches) into total centimeters.

Figure 6.4.1: Program with a function to convert height in feet/inches to centimeters.

```
#include <stdio.h>

/* Converts a height in feet/inches to centimeters */
double HeightFtInToCm(int heightFt, int heightIn) {
    const double CM_PER_IN = 2.54;
    const int IN_PER_FT = 12;
    int totIn;
    double cmVal;

    totIn = (heightFt * IN_PER_FT) + heightIn; // Total inches
    cmVal = totIn * CM_PER_IN; // Conv inch to cm
    return cmVal;
}

int main(void) {
    int userFt;      // User defined feet
    int userIn;      // User defined inches

    // Prompt user for feet/inches
    printf("Enter feet: ");
    scanf("%d", &userFt);

    printf("Enter inches: ");
    scanf("%d", &userIn);

    // Output the conversion result
    printf("Centimeters: %.2f\n",
           HeightFtInToCm(userFt, userIn));

    return 0;
}
```

Enter feet: 5
Enter inches: 8
Centimeters: 172.720000

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Human average height is increasing, attributed to better nutrition. Source: [Our World in Data: Human height](#).

PARTICIPATION ACTIVITY
6.4.1: Mathematical functions.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



Indicate which is a valid use of the HeightFtInToCm() function above. x is type double.

1) `x = HeightFtInToCm(5, 0);`

Valid

Not valid



2) `x = 2 * (HeightFtInToCm(5, 0) + 1.0);`

Valid

Not valid



3) `x = (HeightFtInToCm(5, 0) +`

`HeightFtInToCm(6, 1)) / 2.0;`

Valid

Not valid



4) Suppose `int pow(int y, int z)`

returns y to the power of z. Is the following valid?

`x = pow(2, pow(3, 2));`

Valid

Not valid


zyDE 6.4.1: Temperature conversion.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams

Complete the program by writing and calling a function that converts a temperature from Celsius into Fahrenheit using the formula: $Fahrenheit = \frac{9}{5} * Celsius + 32$

[Load default template...](#)

```

1 #include <stdio.h>
2
3

```

100

Run

```
4 // FINISH Define CelsiusToFahrenheit
5
6
7 int main(void) {
8     double tempF;
9     double tempC;
10
11    printf("Enter temperature in Celsius: ");
12    scanf("%lf", &tempC);
13
14    tempF = 0.0; // FIXME
15
16    printf("Fahrenheit: %lf\n", tempF);
17
18 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Calling functions in expressions

A function call evaluates to the returned value. Thus, a function call often appears within an expression. Ex: `5 + computeSquare(4)` evaluates to $5 + 16$, or 21.

A function with a void return type cannot be used within an expression, instead being used in a statement like: `outputData(x, y);`

PARTICIPATION
ACTIVITY

6.4.2: Function called twice in an expression.



Animation content:

undefined

Animation captions:

1. ComputeSquare() is called within an expression two times, first with the argument 7. 7 is passed to ComputeSquare() the value 49 is returned.
2. ComputeSquare() is called a second time with the argument 9 and the value 81 is returned to the expression.
3. The expression then evaluates to $c2 = 49 + 81$, which assigns variable $c2$ with 130. Lastly, the print statement executes.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION
ACTIVITY

6.4.3: Calls in an expression.



Given the definitions below, which are valid statements?

```
double SquareRoot(double x) { ... }  
void PrintVal(double x) { ... }  
double y;
```

1) `y = SquareRoot(49.0);` 

- Valid
- Invalid

2) `SquareRoot(49.0) = z;` 

- Valid
- Invalid

3) `y = 1.0 + SquareRoot(144.0);` 

- Valid
- Invalid

4) `y =
SquareRoot(SquareRoot(16.0));` 

- Valid
- Invalid

5) `y = SquareRoot();` 

- Valid
- Invalid

6) `SquareRoot(9.0);` 

- Valid
- Invalid

7) `y = PrintVal(9.0);` 

- Valid
- Invalid

8) `PrintVal(9.0);` 

- Valid
- Invalid

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021 

Modular functions for mathematical expressions

Modularity allows more complex functions to incorporate simpler functions. Complex mathematical functions often call other mathematical functions. Ex: A function that calculates the volume or surface area of a cylinder calls a function that returns the area of the cylinder's base, which is needed for both calculations.

Figure 6.4.2: Program that calculates cylinder volume and surface area by calling a modular function for the cylinder's base.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
#include <stdio.h>
#include <math.h>

double CalcCircularBaseArea(double radius) {
    return M_PI * radius * radius;
}

double CalcCylinderVolume(double baseRadius, double height) {
    return CalcCircularBaseArea(baseRadius) * height;
}

double CalcCylinderSurfaceArea(double baseRadius, double height) {
    return (2 * M_PI * baseRadius * height) + (2 *
CalcCircularBaseArea(baseRadius));
}

int main(void) {
    double radius; // User defined feet
    double height; // User defined height

    // Prompt user for radius
    printf("Enter base radius: ");
    scanf("%lf", &radius);

    // Prompt user for height
    printf("Enter height: ");
    scanf("%lf", &height);

    // Output the cylinder volume result
    printf("Cylinder volume: ");
    printf("%.1lf\n", CalcCylinderVolume(radius, height));

    // Output the cylinder surface area result
    printf("Cylinder surface area: ");
    printf("%.3lf\n", CalcCylinderSurfaceArea(radius, height));

    return 0;
}
```

Enter base radius: 10
Enter height: 5
Cylinder volume: 1570.8
Cylinder surface area:
942.478

CHALLENGE ACTIVITY

6.4.1: Function definition: Volume of a pyramid with modular functions.



Define a function CalcPyramidVolume with double data type parameters baseLength, baseWidth, and pyramidHeight, that returns as a double the volume of a pyramid with a rectangular base. CalcPyramidVolume() calls the given CalcBaseArea() function in the

calculation.

Relevant geometry equations:

Volume = base area x height x 1/3

Base area = base length x base width.

(Watch out for integer division).

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 double CalcBaseArea(double baseLength, double baseWidth) {
4     return baseLength * baseWidth;
5 }
6
7 /* Your solution goes here */
8
9 int main(void) {
10     double userLength;
11     double userWidth;
12     double userHeight;
13
14     scanf("%lf", &userLength);
15     scanf("%lf", &userWidth);
16     scanf("%lf", &userHeight);
17
18     printf("Volume: %lf\n", CalcPyramidVolume(userLength, userWidth, userHeight))
```

Run

6.5 Functions with branches/loops

Example: Auction website fee calculator

A function's block of statements may include branches, loops, and other statements. The following example uses a function to compute the fee charged by eBay when a customer sells an item online.

Jacob Adams

UACS100Fall2021

Figure 6.5.1: Function example: Determining fees given an item selling price for an auction website.

```
#include <stdio.h>

/* Returns fee charged by ebay.com given the selling
   price of fixed-price books, movies, music, or
   video-games.
   Fee is $0.50 to list plus a % of the selling price:
   13% for $50.00 or less
   plus 5% for $50.01 to $1000.00
   plus 2% for $1000.01 or more
   Source: http://pages.ebay.com/help/sell/fees.html,
   2012.

   Note: double variables are not normally used for
   dollars/cents
   due to the internal representation's precision, but
   are used
   here for simplicity.
 */

// Function determines eBay price given item selling
price
double CalcEbayFee(double sellPrice) {
    const double BASE_LIST_FEE      = 0.50; // Listing
Fee
    const double PERC_50_OR_LESS    = 0.13; // % $50 or
less
    const double PERC_50_TO_1000    = 0.05; // %
$50.01..$1000.00
    const double PERC_1000_OR_MORE = 0.02; // %
$1000.01 or more
    double feeTotal;                // Resulting
eBay fee

    feeTotal = BASE_LIST_FEE;

    // Determine additional fee based on selling price
    if (sellPrice <= 50.00) { // $50.00 or lower
        feeTotal = feeTotal + (sellPrice *
PERC_50_OR_LESS);
    }
    else if (sellPrice <= 1000.00) { //
$50.01..$1000.00
        feeTotal = feeTotal + (50 * PERC_50_OR_LESS)
        + ((sellPrice - 50) * PERC_50_TO_1000);
    }
    else { // $1000.01 and higher
        feeTotal = feeTotal + (50 * PERC_50_OR_LESS)
        + ((1000 - 50) * PERC_50_TO_1000)
        + ((sellPrice - 1000) * PERC_1000_OR_MORE);
    }

    return feeTotal;
}

int main(void) {
    double sellingPrice;           // User defined selling
price

    // Prompt user for selling price, call eBay fee
function
    printf("Enter item selling price (e.g., 65.00): ");
    scanf("%lf", &sellingPrice);
    printf("eBay fee: $%lf\n",
CalcEbayFee(sellingPrice));

    return 0;
}
```

Enter item selling price (e.g., 65.00):
9.95

eBay fee: \$1.793500

...

Enter item selling price (e.g., 65.00):
40

eBay fee: \$5.700000

...

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

Enter item selling price (e.g., 65.00):
100

eBay fee: \$9.500000

...

Enter item selling price (e.g., 65.00):
500.15

eBay fee: \$29.507500

...

Enter item selling price (e.g., 65.00):
2000

eBay fee: \$74.500000

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

**PARTICIPATION
ACTIVITY**

6.5.1: Analyzing the eBay fee calculator.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021



- 1) For any call to `CalcEbayFee()`, how many assignment statements for the variable `feeTotal` will execute?

Check**Show answer**

- 2) What does `CalcEbayFee()` return if the `sellPrice` argument is 0.0 (show your answer in the form `#.##`)?

Check**Show answer**

- 3) What does `CalcEbayFee()` return if the `sellPrice` argument is 100.00 (show your answer in the form `#.##`)?

Check**Show answer**

Example: Least-common multiple calculator

©zyBooks 12/22/21 12:35 1026490

The following is another example with user-defined functions. The functions keep `main()`'s behavior readable and understandable.

UACS100Fall2021

Figure 6.5.2: User-defined functions make `main()` easy to understand.

```
#include <stdio.h>
#include <stdlib.h>
```

```

// Function prompts user to enter positive non-zero number
int PromptForPositiveNumber(void) {
    int userNum;

    userNum = 0;

    while (userNum <= 0) {
        printf("Enter a positive number (>0): \n");
        scanf("%d", &userNum);

        if (userNum <= 0) {
            printf("Invalid number.\n");
        }
    }

    return userNum;
}

// Function returns greatest common divisor of two inputs
int FindGCD(int aVal, int bVal) {
    int numA;
    int numB;

    numA = aVal;
    numB = bVal;

    while (numA != numB) { // Euclid's algorithm
        if (numB > numA) {
            numB = numB - numA;
        }
        else {
            numA = numA - numB;
        }
    }

    return numA;
}

// Function returns least common multiple of two inputs
int FindLCM(int aVal, int bVal) {
    int lcmVal;

    lcmVal = abs(aVal * bVal) / FindGCD(aVal, bVal);

    return lcmVal;
}

int main(void) {
    int usrNumA;
    int usrNumB;
    int lcmResult;

    printf("Enter value for first input\n");
    usrNumA = PromptForPositiveNumber();

    printf("\nEnter value for second input\n");
    usrNumB = PromptForPositiveNumber();

    lcmResult = FindLCM(usrNumA, usrNumB);

    printf("\nLeast common multiple of %d and %d is %d\n",
          usrNumA, usrNumB, lcmResult);

    return 0;
}

```

Enter value for first input
Enter a positive number (>0):
13

Enter value for second input
Enter a positive number (>0):
7

Least common multiple of 13 and 7 is 91

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION ACTIVITY

6.5.2: Analyzing the least common multiple program.



- 1) Other than main(), which user-defined function calls another user-defined function? Just write the function name.

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 2) How many user-defined function calls exist in the program code?
Note: abs() is not a user-defined function.

Check**Show answer****CHALLENGE ACTIVITY**

6.5.1: Output of functions with branches/loops.



347282.2052980.qx3zqy7

Start

Type the program's output

```
#include <stdio.h>
#include <string.h>

void PrintMessage(char message[]) {
    if (strlen(message) > 10) {
        printf("too long\n");
    } else {
        printf("%s\n", message);
    }
}

int main(void) {
    PrintMessage("You look great!");
    PrintMessage("Where?");

    return 0;
}
```

too long

Where?

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

[Check](#)[Next](#)**CHALLENGE ACTIVITY****6.5.2: Function with branch: Popcorn.**

©zyBooks 12/22/21 12:35 1026490

Jacob Adams



Complete function PrintPopcornTime(), with int parameter bagOunces, and void return type. If bagOunces is less than 3, print "Too small". If greater than 10, print "Too large". Otherwise, compute and print 6 * bagOunces followed by " seconds". End with a newline. Example output for ounces = 7:

42 seconds

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 void PrintPopcornTime(int bagOunces) {
4
5     /* Your solution goes here */
6
7 }
8
9 int main(void) {
10     int userOunces;
11
12     scanf("%d", &userOunces);
13     PrintPopcornTime(userOunces);
14
15     return 0;
16 }
```

[Run](#)**CHALLENGE ACTIVITY****6.5.3: Function with loop: Shampoo.**©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Write a function PrintShampooInstructions(), with int parameter numCycles, and void return type. If numCycles is less than 1, print "Too few.". If more than 4, print "Too many.". Else, print "N: Lather and rinse." numCycles times, where N is the cycle number, followed by "Done.". End with a newline. Example output with input 2:

1: Lather and rinse.

2: Lather and rinse.

Done.

Hint: Declare and use a loop variable.

347282.2052980.qx3zqy7

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
6     int userCycles;
7
8     scanf("%d", &userCycles);
9     PrintShampooInstructions(userCycles);
10
11     return 0;
12 }
```

Run

6.6 Unit testing (functions)

Testing is the process of checking whether a program behaves correctly. Testing a large program can be hard because bugs may appear anywhere in the program, and multiple bugs may interact. Good practice is to test small parts of the program individually, before testing the entire program, which can more readily support finding and fixing bugs. **Unit testing** is the process of individually testing a small part or unit of a program, typically a function. A unit test is typically conducted by creating a **testbench**, a.k.a. test harness, which is a separate program whose sole purpose is to check that a function returns correct output values for a variety of input values. Each unique set of input values is known as a **test vector**.

Consider a function HrMinToMin() that converts time specified in hours and minutes to total minutes. The figure below shows a test harness that tests that function. The harness supplies various input vectors like (0,0), (0,1), (0,99), (1,0), etc. Note: Assertions need to be enabled as virtual machine arguments on the command line or in your IDE.

Figure 6.6.1: Test harness for the function HrMinToMin().

```
#include <stdio.h>

int HrMinToMin(int origHours, int origMinutes) {
    int totMinutes; // Resulting minutes

    totMinutes = (origHours * 60) + origMinutes;

    return origMinutes;
}

int main(void) {
    printf("Testing started\n");

    printf("0:0, expecting 0, got: %lf\n", HrMinToMin(0,0));
    printf("0:1, expecting 1, got: %lf\n", HrMinToMin(0,1));
    printf("0:99, expecting 99, got: %lf\n", HrMinToMin(0,99));
    printf("1:0, expecting 60, got: %lf\n", HrMinToMin(1,0));
    printf("5:0, expecting 300, got: %lf\n", HrMinToMin(5,0));
    printf("2:30, expecting 150, got: %lf\n", HrMinToMin(2,30));
    // Many more test vectors would be typical...

    printf("Testing completed\n");

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Testing started
 0:0, expecting 0, got: 0.000000
 0:1, expecting 1, got: 1.000000
 0:99, expecting 99, got: 99.000000
 1:0, expecting 60, got: 0.000000
 5:0, expecting 300, got: 0.000000
 2:30, expecting 150, got: 30.000000
 Testing completed

Manually examining the program's printed output reveals that the function works for the first several vectors, but fails on the next several vectors, highlighted with colored background. Examining the output, one may note that the output minutes is the same as the input minutes; examining the code indeed leads to noticing that parameter origMinutes is being returned rather than variable totMinutes. Returning totMinutes and rerunning the test harness yields correct results.

Each bug a programmer encounters can improve a programmer by teaching him/her to program differently, just like getting hit a few times by an opening door teaches a person not to stand near a closed door.

PARTICIPATION ACTIVITY
6.6.1: Unit testing.


- 1) A test harness involves temporarily modifying an existing program to test a particular function within that program.

- True
- False

- 2) Unit testing means to modify function inputs in small steps known as units.



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



- True
 False

Manually examining a program's printed output is cumbersome and error prone. A better test harness would only print a message for incorrect output. The language provides a compact way to print an error message when an expression evaluates to false. `assert()` is a macro (similar to a function) that prints an error message and exits the program if `assert()`'s input expression is false. The error message includes the current line number and the expression (a nifty trick enabled by using a macro rather than an actual function; details are beyond our scope). Using `assert` requires first including the `assert` library, part of the standard library, as shown below.

Figure 6.6.2: Test harness with assert for the function `HrMinToMin()`.

```
#include <stdio.h>
#include <assert.h>

double HrMinToMin(int origHours, int origMinutes) {
    int totMinutes; // Resulting minutes

    totMinutes = (origHours * 60) + origMinutes;

    return origMinutes;
}

int main(void) {
    printf("Testing started\n");

    assert(HrMinToMin(0, 0) == 0);
    assert(HrMinToMin(0, 1) == 1);
    assert(HrMinToMin(0, 99) == 99);
    assert(HrMinToMin(1, 0) == 60);
    assert(HrMinToMin(5, 0) == 300);
    assert(HrMinToMin(2, 30) == 150);
    // Many more test vectors would be typical...

    printf("Testing completed\n");

    return 0;
}
```

```
Testing started
Assertion failed: (HrMinToMin(1, 0) == 60), function main, file main.c, line 19.
```

Using branches for unit tests

If you have studied branches, you may recognize that each `assert` statement in `main()` could be replaced by an `if` statement like:

```
if ( HrMinToMin(0, 0) != 0 ) {  
    printf("0:0, expecting 0, got: %lf", HrMinToMin(0, 0));  
}
```

But the assert is more compact.

assert() enables compact readable test harnesses, and also eases the task of examining the program's output for correctness; a program without detected errors would simply output "Testing started" followed by "Testing completed".

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

A programmer should choose test vectors that thoroughly exercise a function. Ideally the programmer would test all possible input values for a function, but such testing is simply not practical due to the large number of possibilities -- a function with one integer input has over 4 billion possible input values, for example. Good test vectors include a number of normal cases that represent a rich variety of typical input values. For a function with two integer inputs as above, variety might include mixing small and large numbers, having the first number large and the second small (and vice-versa), including some 0 values, etc. Good test vectors also include **border cases** that represent fringe scenarios. For example, border cases for the above function might include inputs 0 and 0, inputs 0 and a huge number like 9999999 (and vice-versa), two huge numbers, a negative number, two negative numbers, etc. The programmer tries to think of any extreme (or "weird") inputs that might cause the function to fail. For a simple function with a few integer inputs, a typical test harness might have dozens of test vectors. For brevity, the above examples had far fewer test vectors than typical.

PARTICIPATION
ACTIVITY

6.6.2: Assertions and test cases.



- 1) Using assert() is a preferred way to test a function.

- True
 False



- 2) For a function, border cases might include 0, a very large negative number, and a very large positive number.

- True
 False



- 3) For a function with three integer inputs, about 3-5 test vectors is likely sufficient for testing purposes.

- True
 False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



- 4) A good programmer takes the time to



test all possible input values for a function.

- True
- False

Exploring further:

- [assert reference page](#) from cplusplus.com

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

CHALLENGE ACTIVITY

6.6.1: Unit testing.



Add two more statements to main() to test inputs 3 and -1. Use print statements similar to the existing one (don't use assert).

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 // Function returns origNum cubed
4 int CubeNum(int origNum) {
5     return origNum * origNum * origNum;
6 }
7
8 int main(void) {
9
10    printf("Testing started\n");
11
12    printf("2: Expecting 8, got: %d\n", CubeNum(2));
13
14    /* Your solution goes here */
15
16    printf("Testing completed\n");
17
18    return 0;
}
```

Run

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

6.7 How functions work

Each function call creates a new set of local variables, forming part of what is known as a **stack frame**. A return causes those local variables to be discarded.

PARTICIPATION ACTIVITY

6.7.1: Function calls and returns.



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Animation captions:

1. Each function call creates a new set of local variables.
2. Each return causes those local variables to be discarded.

Some knowledge of how a function call and return works at the assembly level can not only satisfy curiosity, but can also lead to fewer mistakes when parameter and return items become more complex. The following animation illustrates by showing, for a function named FindMax(), some sample high-level code, compiler-generated assembly instructions in memory, and data in memory during runtime. This animation presents advanced material intended to provide insight and appreciation for how a function call and return works.

The compiler generates instructions to copy arguments to parameter local variables, and to store a return address. A jump instruction jumps from main to the function's instructions. The function executes and stores results in a designated return value location. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

PARTICIPATION ACTIVITY

6.7.2: How function call/return works.

**Animation content:****undefined****Animation captions:**

1. The compiler converts high-level code into assembly instructions in memory.
2. Before executing the function, arguments are copied to parameter local variables and a return address is stored.
3. The function executes and stores the result in a designated return value location.
4. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION ACTIVITY

6.7.3: How functions work.





1) After a function returns, its local variables keep their values, which serve as their initial values the next time the function is called.

- True
- False

2) A return address indicates the value returned by the function.

- True
- False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



6.8 Functions: Common errors

A common error is to copy-and-paste code among functions but then not complete all necessary modifications to the pasted code. For example, a programmer might have developed and tested a function to convert a temperature value in Celsius to Fahrenheit, and then copied and modified the original function into a new function to convert Fahrenheit to Celsius as shown:

Figure 6.8.1: Copy-paste common error: Pasted code not properly modified. Find error on the right.

```
double Cel2Fah(double celVal) {  
    double convTmp;  
    double fahVal;  
  
    convTmp = (9.0 / 5.0) * celVal;  
    fahVal = convTmp + 32;  
  
    return fahVal;  
}  
  
double Fah2Cel(double fahVal) {  
    double convTmp;  
    double celVal;  
  
    convTmp = fahVal - 32;  
    celVal = convTmp * (5.0 / 9.0);  
  
    return celVal;  
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

The programmer forgot to change the return statement to return celVal rather than fahVal. Copying-and-pasting code is a common and useful time-saver, and can reduce errors by starting with known-correct code. Our advice is that when you copy-paste code, be extremely vigilant in making all necessary modifications. Just as the awareness that dark alleys or wet roads may be dangerous can cause you to vigilantly observe your surroundings or drive carefully, the awareness that copying-and-pasting is a common source of errors, may cause you to more vigilantly ensure you modify a pasted function correctly.

PARTICIPATION ACTIVITY

6.8.1: Copy-pasted sum-of-squares code.



Original parameters were num1, num2, num3. Original code was:

```
int sum;  
  
sum = (num1 * num1) + (num2 * num2) + (num3 * num3);  
  
return sum;
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

New parameters are num1, num2, num3, num4. Find the error in the copy-pasted new code below.

1)

```
int sum;  
  
sum =  
(num1 * num1) + (num2 * num2) + (num3 * num3)  
+ (num3 * num4)  
;  
  
return sum;
```



Another common error is to return the wrong variable, such as typing `return convTmp;` instead of `fahVal` or `celVal`. The function will work and sometimes even return the correct value.

Failing to return a value for a function is another common error. If execution reaches the end of a function's statements, the function automatically returns. For a function with a void return type, such an automatic return poses no problem, although some programmers recommend including a return statement for clarity. But for a function defined to return a value, the returned value is undefined; the value could be anything. For example, the user-defined function below lacks a return statement:

Figure 6.8.2: Missing return statement common error: Program may sometimes work, leading to hard-to-find bug.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
#include <stdio.h>

int StepsToFeet(int baseSteps) {
    const int FEET_PER_STEP = 3; // Unit conversion
    int feetTot; // Corresponding feet to steps

    feetTot = baseSteps * FEET_PER_STEP;
}

int main(void) {
    int stepsInput; // User defined steps
    int feetTot; // Corresponding feet to steps

    // Prompt user for input
    printf("Enter number of steps walked: ");
    scanf("%d", &stepsInput);

    // Call functions to convert steps to feet
    feetTot = StepsToFeet(stepsInput);
    printf("Feet: %d\n", feetTot);

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
 Enter number of steps walked: 1000
 Jacob Adams
 Feet: 3000 UACS100Fall2021

Sometimes a function with a missing return statement (or just `return;`) still returns the correct value. The reason is that the compiler uses a memory location to return a value to the calling expression. That location may have also been used by the compiler to store a local variable of that function. If that local variable happens to be the item that was supposed to be returned, the value in that location is the correct return value. But a later seemingly unrelated change to a function, like defining a new variable, may cause the compiler to use different memory locations, and the function suddenly no longer returns the correct value, leading to a bewildered programmer.

PARTICIPATION ACTIVITY

6.8.2: Common function errors.



Find the error in the function's code.

```
1) ComputeSumOfSquares(int num1, int num2) {
    int sum;

    sum = (num1 * num1) + (num2 * num2)
;

    return;
```

©zyBooks 12/22/21 12:35 1026490
 Jacob Adams
 UACS100Fall2021



}

```
int ComputeEquation1(int num, int val, int k  
2) ) {  
    int sum;  
  
    sum = (num * val) + (k * val)  
;  
    return num;  
}
```



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION ACTIVITY

6.8.3: Common function errors.



- 1) Forgetting to return a value from a function is a common error.
 True
 False
- 2) Copying-and-pasting code can lead to common errors if all necessary changes are not made to the pasted code.
 True
 False
- 3) Returning the incorrect variable from a function is a common error.
 True
 False
- 4) Is this function correct for squaring an integer?

```
int sqr(int a) {  
    int t;  
    t = a * a;  
}
```


 Yes
 No
- 5) Is this function correct for squaring an



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

integer?

```
int sqr(int a) {
    int t;
    t = a * a;
    return a;
}
```

- Yes
- No

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

CHALLENGE ACTIVITY

6.8.1: Function errors: Copying one function to create another.



Using the CelsiusToKelvin function as a guide, create a new function, changing the name to KelvinToCelsius, and modifying the function accordingly.

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 double CelsiusToKelvin(double valueCelsius) {
4     double valueKelvin;
5
6     valueKelvin = valueCelsius + 273.15;
7
8     return valueKelvin;
9 }
10
11 /* Your solution goes here */
12
13 int main(void) {
14     double valueC;
15     double valueK;
16
17     valueC = 10.0;
18     printf("%lf C is %lf K\n", valueC, CelsiusToKelvin(valueC));
```

Run

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

6.9 Pass by pointer

Pass by pointer

New programmers sometimes assign a value to a parameter, believing the assignment updates the corresponding argument variable. An example situation is when a function should return two values,

whereas a function's *return* construct can only return one value. Assigning a normal parameter fails to update the argument's variable, because normal parameters are **pass by value**, meaning the argument's value is copied into a local variable for the parameter.

PARTICIPATION ACTIVITY

6.9.1: Assigning a normal pass by value parameter has no impact on the corresponding argument.



©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021**Animation captions:**

1. The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin.
2. ConvHrMin's parameters are passed by value, so the arguments' values are copied into local variables.
3. Upon return, ConvHrMin's local variables are discarded. hrVal and minVal are local copies that do not impact usrHr and usrMin.

In contrast, defining a parameter as pointer enables updating of an argument variable. The calling function passes a pointer to a variable by prepending &, and each access in the function dereferences that pointer by prepending *, thus referring to the argument variable's memory location.

PARTICIPATION ACTIVITY

6.9.2: A pass by reference parameter allows a function to update an argument variable.

**Animation captions:**

1. The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin.
2. The & indicates that the function passes a pointer to the hrVal and minVal parameters.
3. Parameters passed by reference refer to that variable's memory location, so updates to hrVal and minVal update usrHr and usrMin.
4. Upon return from ConvHrMin, usrHr and usrMin retain the updated values.

Defining a parameter as a pointer to enable updating the argument variable is commonly known as **pass by reference**. However, to avoid confusion with pass by reference in C++, which is achieved quite differently, this material uses the term **pass by pointer**. Pointers are introduced elsewhere; here, the programmer need only remember to pass the argument variable using &, and to define and access the parameter using *.

Pass by pointer parameters should be used sparingly. For the case of two return values, commonly a programmer should instead create two functions. For example, defining two separate functions `int StepsToFeet(int baseSteps)` and `int StepsToCalories(int totCalories)` is better than a single function

`void StepsToFeetAndCalories(int baseSteps, int* baseFeet, int* totCalories)`. The separate functions support modular development, and enables use of the functions in an expression as in `if (StepsToFeet(mySteps) < 100)`.

Using multiple pass by pointer parameters makes sense when the output values are intertwined, such as computing monetary change, whose function might be

`void ComputeChange(int totCents, int* numQuarters, int* numDimes, int* numNickels)`

or converting from polar to Cartesian coordinates, whose function might be

`void PolarToCartesian(int radialPol, int anglePol, int* xCar, int* yCar)`.

Jacob Adams
UACS100Fall2021

zyDE 6.9.1: Calculating monetary change.

Complete the monetary change program. Use the fewest coins (i.e., using maximum large coins first).

Load default template

```
1 #include <stdio.h>
2
3 // FIXME: Add parameters for dimes, nickels, and pennies.
4 void ComputeChange(int totCents, int* numQuarters ) {
5
6     printf("FIXME: Finish writing ComputeChange\n");
7
8     *numQuarters = totCents / 25;
9 }
10
11 int main(void) {
12     int userCents;
13     int numQuarters;
14     // FIXME add variables for dimes, nickels, pennies
15
16
17     printf("Enter total cents: \n");
18 }
```

83

Run

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION ACTIVITY

6.9.3: Function definition returns and arguments.



Choose the most appropriate function definition.

- 1) Convert inches into centimeters.

- void InchToCM(double inches, double centimeters) ...
- double InchToCM(double inches) ...
- More than one function should be written.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 2) Compute the area and diameter of a circle given the radius.

- void GetCircleAreaDiam(double radius, double* area, double* diameter) ...
- double GetCircleAreaDiam(double radius, double* area) ...
- double, double GetCircleAreaDiam(double radius) ...
- More than one function should be written.

PARTICIPATION ACTIVITY

6.9.4: Function with pass by pointer.

Given the following ConvDigits() function,

```
void ConvDigits(int inNum, int* tensVal, int* onesVal) {  
    *onesVal = inNum % 10;  
    *tensVal = inNum / 10;  
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 1) What is the value of the variable numTens, after the following function call?

```
int numTens;  
int numOnes;  
  
ConvDigits(45, &numTens, &numOnes);
```

Check**Show answer**

- 2) What is the value of the variable numOnes, after the following function call?

```
int numTens;  
int numOnes;  
  
ConvDigits(93, &numTens, &numOnes);
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Check**Show answer**

- 3) Provide the code needed to obtain a pointer to a variable named tensDigit.

Check**Show answer**

- 4) Complete the function call to ConvDigits() to store the tens place and ones place digits of the number 32 within the integer variables tensDigit and onesDigit.

```
ConvDigits (  ) ;
```

Check**Show answer**

Avoid assigning pass by value parameters

Although a pass by value parameter creates a local copy, good practice is to avoid assigning such a parameter. The following code is correct but bad practice.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Figure 6.9.1: Programs should not assign pass by value parameters.

```
int IntMax(int numVal1, int numVal2) {
    if (numVal1 > numVal2) {
        numVal2 = numVal1; // numVal2 holds max
    }

    return numVal2;
}
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

Assigning a parameter can reduce code slightly, but is widely considered a lazy programming style. Assigning a parameter can mislead a reader into believing the argument variable is supposed to be updated. Assigning a parameter also increases likelihood of a bug caused by a statement reading the parameter later in the code but assuming the parameter's value is the original passed value.

PARTICIPATION ACTIVITY

6.9.5: Assigning a pass by value parameter.



- 1) Assigning a pass by value parameter in a function is discouraged due to potentially confusing a program reader into believing the argument is being updated.

- True
 False



- 2) Assigning a pass by value parameter in a function is discouraged due to potentially leading to a bug where a later line of code reads the parameter assuming the parameter still contains the original value.

- True
 False



- 3) Assigning a pass by value parameter can avoid having to declare an additional local variable.

- True
 False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Exploring further:

- [Passing arguments by value and by reference](#) from msdn.microsoft.com

**CHALLENGE
ACTIVITY****6.9.1: Function pass by pointer: Transforming coordinates.**

Define a function CoordTransform() that transforms its first two input parameters xVal and yVal into two output parameters xValNew and yValNew. The function returns void. The transformation is new = (old + 1) * 2. Ex: If xVal = 3 and yVal = 4, then xValNew is 8 and yValNew is 10.

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
6     int xValNew;
7     int yValNew;
8     int xValUser;
9     int yValUser;
10
11    scanf("%d", &xValUser);
12    scanf("%d", &yValUser);
13
14    CoordTransform(xValUser, yValUser, &xValNew, &yValNew);
15    printf("(%d, %d) becomes (%d, %d)\n", xValUser, yValUser, xValNew, yValNew);
16
17    return 0;
18 }
```

Run

◀ ▶

◀ ▶

6.10 Functions with array parameters

Functions commonly have array parameters. The following program uses a function to calculate the average value of the elements within an array of test scores.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Figure 6.10.1: Array parameters in an average test score calculation program.

```
#include <stdio.h>

double CalculateAverage(const double scoreVals[], int numVals) {
    int i;
    double scoreSum = 0.0;

    for (i = 0; i < numVals; ++i) {
        scoreSum = scoreSum + scoreVals[i];
    }

    return scoreSum / numVals;
}

int main(void) {
    const int NUM_SCORES = 4;      // Array size
    double testScores[NUM_SCORES]; // User test scores
    int i;
    double averageScore;

    // Prompt user to enter test scores
    printf("Enter %d test scores:\n", NUM_SCORES);
    for (i = 0; i < NUM_SCORES; ++i) {
        printf("Test score: ");
        scanf("%lf", &(testScores[i]));
    }
    printf("\n");

    // Call function to calculate average
    averageScore = CalculateAverage(testScores, NUM_SCORES);
    printf("Average adjusted test score: ");
    printf("%lf\n", averageScore);

    return 0;
}
```

Enter 4 test scores:

Test score: 90.5

Test score: 92.0

Test score: 87.5

Test score: 97.0

Average adjusted test score:

91.750000

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

The parameter definition (yellow highlighted) uses [] to indicate an array parameter. The function call's argument (orange highlighted) does not use []. The compiler automatically passes array arguments by passing a pointer to the array, rather than making a copy like for other parameter types, in part to avoid the inefficiency of copying large arrays.

Functions with array parameters, other than string parameters (discussed elsewhere), need a second parameter indicating the number of elements in the array. Ex: numVals indicates the number of elements within the scoresVals array.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

PARTICIPATION ACTIVITY
6.10.1: Functions with array parameters.


- 1) Passing an array creates a copy of that array within the function.

 True


False

- 2) An array is automatically passed by pointer.

 True False

- 3) In the GetAvgScore function, totalScores must be an explicit pointer parameter for an array.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
void GetAvgScore(int* totalScores, int*  
    numScores) {  
    // ...  
}
```

 True False**PARTICIPATION ACTIVITY**

6.10.2: Functions with arrays.



- 1) Complete the FindMin function definition to have an array parameter of type double named arrayVals and second parameter of type int numVals indicating the number of array elements.

```
double FindMin (
```

```
    ) {  
        // ...  
    }
```

Check**Show answer**

- 2) Complete the PrintVals function to print each array element on a separate line.



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```

void PrintVals(int
arrayVals[], int numElements)
{
    int i;

    for(i = 0; i <
        ; ++i) {
        printf("%d\n",
arrayVals[i]);
    }
}

```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Check**Show answer**

- 3) sellingPrices is an array of type double with 10 elements.

Complete the statement to find the lowest price by calling the GetLowestPrice function defined below.

```

GetLowestPrice(double itemPrices[],
int numItems);

lowestPrice = GetLowestPrice(
);

```

Check**Show answer**

A programmer can explicitly define an array parameter as a pointer, but good practice is to use [] to define array parameters to clearly indicate the parameter is an array and not a pointer to a single variable. Ex: **void PrintVals(int arrVals[], int numVals)** is equivalent to **void PrintVals(int* arrVals, int numVals)**.

Because an array is passed to a function by passing a pointer to the array, a function can modify the elements of an array argument.

PARTICIPATION ACTIVITY

6.10.3: Functions can modify elements of an array argument.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Animation captions:

1. Arrays are automatically passed by pointer. So the address of testScores is passed to the scoreVals parameter.
2. A function can modify the elements of an array argument. Because scoreVals is a pointer to testScores, the function modifies testScores' elements.
3. Function adds 2.0 to all elements of the array argument.

The keyword **const** can be prepended to a function's array parameter to prevent the function from modifying the parameter. In the following program, the AdjustScores function modifies the array so defines a normal array parameter (highlighted yellow). The PrintScores and CalculateAverage functions do *not* modify the array so define a const array parameter (highlighted orange).

Figure 6.10.2: Normal and const array parameters in test score adjustment and averaging program.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```

#include <stdio.h>

void AdjustScores(double scoreVals[], int numVals,
                  double scoreAdj) {
    int i;

    for (i = 0; i < numVals; ++i) {
        scoreVals[i] = scoreVals[i] + scoreAdj;
    }
}

void PrintScores(const double scoreVals[], int numVals) {
    int i;

    for (i = 0; i < numVals; ++i) {
        printf(" %lf", scoreVals[i]);
    }
    printf("\n");
}

double CalculateAverage(const double scoreVals[], int numVals) {
    int i;
    double scoreSum = 0;

    for (i = 0; i < numVals; ++i) {
        scoreSum = scoreSum + scoreVals[i];
    }

    return scoreSum / numVals;
}

int main(void) {
    const int NUM_SCORES = 4;      // Array size
    double testScores[NUM_SCORES]; // User test scores
    int i;
    double averageScore;

    // Prompt user to enter test scores
    printf("Enter %d test scores:\n", NUM_SCORES);
    for (i = 0; i < NUM_SCORES; ++i) {
        printf("Test score: ");
        scanf("%lf", &(testScores[i]));
    }
    printf("\n");

    // Print original scores
    printf("Original test scores: ");
    PrintScores(testScores, NUM_SCORES);

    AdjustScores(testScores, NUM_SCORES, 2.0);

    printf("Adjusted test scores: ");
    PrintScores(testScores, NUM_SCORES);

    // Call function to calculate average
    averageScore = CalculateAverage(testScores, NUM_SCORES);
    printf("Average adjusted test score: ");
    printf("%lf\n", averageScore);

    return 0;
}

```

©zyBooks 12/22/21 12:35 1026490
 Jacob Adams
 UACS100Fall2021

©zyBooks 12/22/21 12:35 1026490
 Jacob Adams
 UACS100Fall2021

```
Enter 4 test scores:
```

```
Test score: 90.0
```

```
Test score: 95.5
```

```
Test score: 97.0
```

```
Test score: 92.5
```

```
Original test scores: 90.000000 95.500000 97.000000 92.500000
```

```
Adjusted test scores: 92.000000 97.500000 99.000000 94.500000
```

```
Average adjusted test score: 95.750000
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

PARTICIPATION ACTIVITY**6.10.4: Const array function parameters.**

Can the following functions modify the array parameter?

1) `void PrintArray(double userNums[], int numElements)`

- Yes
- No



2) `int GetHighScore(int userScores[], const int numHighScores)`

- Yes
- No



3) `int FindIndex(const int searchVals[], int numVal, int key)`

- Yes
- No



©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

CHALLENGE ACTIVITY**6.10.1: Functions with array parameters.**

347282.2052980.qx3zqy7

Start

Type the program's output

```
#include <stdio.h>

void PrintScoreVals(const int scoreVals[], int numVals) {
    int i;

    for (i = 0; i < numVals; ++i) {
        printf("%d\n", scoreVals[i]);
    }
}

int main(void) {
    const int NUM_SCORES = 3;
    int quizScores[NUM_SCORES];
    int i;

    for (i = 0; i < NUM_SCORES; ++i) {
        scanf("%d", &(quizScores[i]));
    }

    PrintScoreVals(quizScores, NUM_SCORES);

    return 0;
}
```

Input

7

2

8

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Output

7

2

8

1

2

3

Check

Next

CHALLENGE ACTIVITY

6.10.2: Modify an array parameter.



Write a function SwapArrayEnds() that swaps the first and last elements of the function's array parameter. Ex: sortArray = {10, 20, 30, 40} becomes {40, 20, 30, 10}.

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2
3 /* Your solution goes here */
4
5 int main(void) {
6     const int SORT_ARR_SIZE = 4;
7     int sortArray[SORT_ARR_SIZE];
8     int i;
9     int userNum;
10
11    for (i = 0; i < SORT_ARR_SIZE; ++i) {
12        scanf("%d", &sortArray[i]);
13    }
14
15    SwapArrayEnds(sortArray, SORT_ARR_SIZE);
16
17    for (i = 0; i < SORT_ARR_SIZE; ++i) {
18        printf("%d ", sortArray[i]);
19    }
20}
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
17     for (i = 0; i < SORT_ARRAY_SIZE; ++i) {
18         printf("%d ", sortArray[i]);
```

Run

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

6.11 Functions with C string parameters

Functions commonly modify strings. The following function modifies a string by replacing spaces with hyphens.

Figure 6.11.1: Modifying a string parameter.

```
#include <stdio.h>
#include <string.h>

// Function replaces spaces with hyphens
void StrSpaceToHyphen(char modString[]) {
    int i; // Loop index

    for (i = 0; i < strlen(modString); ++i) {
        if (modString[i] == ' ') {
            modString[i] = '-';
        }
    }
}

int main(void) {
    const int INPUT_STR_SIZE = 50; // Input string size
    char userStr[INPUT_STR_SIZE]; // Input string from
    user

    // Prompt user for input
    printf("Enter string with spaces: \n");
    fgets(userStr, INPUT_STR_SIZE, stdin);

    // Call function to modify user defined string
    StrSpaceToHyphen(userStr);

    printf("String with hyphens: %s\n", userStr);

    return 0;
}
```

Enter string with spaces:
Hello there everyone.
String with hyphens: Hello-there-
everyone.
...
Enter string with spaces:
Good bye now !!!
String with hyphens: Good-bye--now--
-!!!

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

The parameter definition (yellow highlighted) uses [] to indicate an array parameter. The function call's argument (orange highlighted) does not use []. The compiler *automatically passes the string as a pointer*. Hence, the above function modifies the original string argument (userStr) and not a copy.

The `strlen()` function can be used to determine the length of the string argument passed to the function. So, unlike functions with array parameters of other types, a function with a string parameter does not require a second parameter to specify the string size.

zyDE 6.11.1: Modifying a string parameter: Spaces to hyphens.

1. Run the program, noting correct output.
2. Modify the function to also replace each '!' by a '?'.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021


The screenshot shows a code editor window titled "Load default template...". The code is as follows:

```

1 #include <stdio.h>
2 #include <string.h>
3
4 // Function replaces spaces with hyp
5 void StrSpaceToHyphen(char modString
6     int i; // Loop index
7
8     for (i = 0; i < strlen(modString)
9         if (modString[i] == ' ') {
10             modString[i] = '-';}
11         }
12     }
13 }
14
15 }
16
17 int main(void) {
18

```

On the right, there is a "Run" button and an output window displaying "Hello there everyone!!!".

Recall that a `scanf()` string argument did not require a prepended & as in `scanf("%s", userName);`, in contrast to integer or other argument types. The & is not needed because the compiler automatically passes an array as a pointer. The same is true for the `userStr` argument in `fgets()` above.

PARTICIPATION ACTIVITY

6.11.1: Functions with string parameters.



- 1) A parameter declared as `char`

`movieTitle[]` is a string.

- True
- False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

- 2) For a function with a string parameter,
the function must include a second
parameter for the string size.



True False

- 3) To pass a string to a function, the argument must include [], as in `GetMovieRating(favMovie[])`.

 True False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



A programmer can explicitly define an array parameter as a pointer. The following uses `char* modString` instead of the earlier `char modString[]`. Such pointer parameters are common for string parameters, such as in the string library functions.

Figure 6.11.2: Modifying a string using a pointer parameter.

```
#include <stdio.h>
#include <string.h>

// Function replaces spaces with hyphens
void StrSpaceToHyphen(char* modString) {
    int i; // Loop index

    for (i = 0; i < strlen(modString); ++i) {
        if (modString[i] == ' ') {
            modString[i] = '-';
        }
    }
}

int main(void) {
    const int INPUT_STR_SIZE = 50; // Input string size
    char userStr[INPUT_STR_SIZE]; // Input string from
    user

    // Prompt user for input
    printf("Enter string with spaces: \n");
    fgets(userStr, INPUT_STR_SIZE, stdin);

    // Call function to modify user defined string
    StrSpaceToHyphen(userStr);

    printf("String with hyphens: %s\n", userStr);

    return 0;
}
```

Enter string with spaces:
Hello there everyone!
String with hyphens: Hello-there-
everyone!

...

Enter string with spaces:
Good bye now !!!
String with hyphens: Good-bye--now--
-!!!

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



PARTICIPATION ACTIVITY

6.11.2: Functions with string parameters.



- 1) Passing a string to a function creates a

copy of that string within the function.

- True
 - False
- 2) A string is automatically passed by pointer.
- True
 - False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

CHALLENGE ACTIVITY

6.11.1: Use an existing function.

Use the function GetUserInfo() to get a user's information. If user enters 20 and Holly, sample program output is:

Holly is 20 years old.

347282.2052980.qx3zqy7

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void GetUserInfo(int* userAge, char userName[]) {
5     printf("Enter your age: \n");
6     scanf("%d", userAge);
7     printf("Enter your name: \n");
8     scanf("%s", userName);
9 }
10
11 int main(void) {
12     int userAge;
13     char userName[30];
14
15     /* Your solution goes here */
16
17     printf("%s is %d years old.\n", userName, userAge);
18 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Run

CHALLENGE ACTIVITY

6.11.2: Modify a string parameter.

Complete the function to replace any period by an exclamation point. Ex: "Hello. I'm Miley. Nice to meet you." becomes:

"Hello! I'm Miley! Nice to meet you!"

347282.2052980.qx3zqy7

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void MakeSentenceExcited(char* sentenceText) {
5     /* Your solution goes here */
6 }
7
8 int main(void) {
9     const int TEST_STR_SIZE = 50;
10    char testStr[TEST_STR_SIZE];
11
12    fgets(testStr, TEST_STR_SIZE, stdin);
13    MakeSentenceExcited(testStr);
14    printf("%s", testStr);
15
16    return 0;
17}
```

Run

6.12 Functions with array parameters: Common errors

For arrays other than C strings, a common error is defining a function with an array parameter without a second parameter indicating the number of array elements. Without a parameter indicating the number of array elements, the function will only work for one specific array size. A programmer may call the function with a larger or smaller array, which may lead to incorrect results.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

PARTICIPATION
ACTIVITY

6.12.1: Common error: Functions with arrays without a parameter indicating the number of array elements.



Animation captions:

1. Passing the 7 element array to the GetAvgScore will incorrectly calculate the average of the first 5 elements, not all 7 elements. GetAvgScore does not use a second parameter specifying the number of array elements.
2. For an array with less than 5 elements, the function will access memory outside the array bounds.

PARTICIPATION ACTIVITY

6.12.2: Function with array missing parameter indicating number of array elements.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



- 1) The GetMinScore() function has a common error of not including a parameter to indicate the number of array elements. Given the arrays preGames, seasonGames, and playoffGames, which function call does not return the minimum value?

```
int GetMinScore(int scoreVals[]) {  
    int i;  
    int minScore;  
  
    minScore = scoreVals[0];  
    for (i = 0; i < 5; ++i) {  
        if (scoreVals[i] < minScore) {  
            minScore = scoreVals[i];  
        }  
    }  
  
    return minScore;  
}  
  
int preGames[]      = { 80, 45, 86, 85,  
64, 99, 98 };  
int seasonGames[]  = { 92, 90, 84, 82,  
98 };  
int playoffGames[] = { 85, 83, 75, 90,  
70, 68, 69 };
```

- GetMinScore(preGames);
 - GetMinScore(seasonGames);
 - GetMinScore(playoffGames);
- 2) Which function definition correctly supports arrays of any size?

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



```
int GetMaxScoreA(int scoreVals[]) {  
    int i;  
    int maxScore;  
  
    maxScore = scoreVals[0];  
    for (i = 0; i < sizeof(scoreVals);  
        ++i) {  
        if (scoreVals[i] > maxScore) {  
            maxScore = scoreVals[i];  
        }  
    }  
  
    return maxScore;  
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
int GetMaxScoreB(int scoreVals[], int  
numScores) {  
    int i;  
    int maxScore;  
  
    maxScore = scoreVals[0];  
    for (i = 0; i < numScores; ++i) {  
        if (scoreVals[i] > maxScore) {  
            maxScore = scoreVals[i];  
        }  
    }  
  
    return maxScore;  
}
```

- GetMaxScoreA()
- GetMaxScoreB()

Another common error is modifying array elements in a function that should not modify the array. Arrays are automatically passed to functions using a pointer to the array. So, a function that modifies an array parameter will modify the array argument passed to the function, and not a local copy.

The `FindMaxAbsValueIncorrect()` function incorrectly modifies the `inputVals` array to find the array element with the largest absolute value.

Figure 6.12.1: `FindMaxAbsValueIncorrect()` incorrectly modifies `inputVals` array to find element with the largest absolute value.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```

#include <stdio.h>
#include <math.h>

float FindMaxAbsValueIncorrect(float inputVals[], int numVals) {
    int i;
    float maxAbsVal;

    // Incorrectly updates inputVals to calculate absolute value
    // of array elements
    for (i = 0; i < numVals; ++i) {
        inputVals[i] = fabs(inputVals[i]);
    }

    maxAbsVal = inputVals[0];
    for (i = 0; i < numVals; ++i) {
        if (inputVals[i] > maxAbsVal) {
            maxAbsVal = inputVals[i];
        }
    }

    return maxAbsVal;
}

int main(void) {
    const int NUM_VALUES = 5;
    // Array of changes in temperatures
    float tempChanges[5] = {10.0, 0.5, -5.1, -11.2, 3.0};
    float maxAbsChange;
    int i;

    // Print array before function call
    printf("tempChanges array before function call: ");
    for(i = 0; i < NUM_VALUES; ++i) {
        printf("%f ", tempChanges[i]);
    }
    printf("\n");

    // Find the largest temperature change, and print result.
    maxAbsChange = FindMaxAbsValueIncorrect(tempChanges, NUM_VALUES);
    printf("Max absolute temperature change: %f\n", maxAbsChange);

    // Print array after function call
    printf("tempChanges array after function call: ");
    for(i = 0; i < NUM_VALUES; ++i) {
        printf("%f ", tempChanges[i]);
    }
    printf("\n");

    return 0;
}

```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```

tempChanges array before function call: 10.000000 0.500000 -5.100000 -11.200000 3.000000
Max absolute temperature change: 11.200000
tempChanges array after function call: 10.000000 0.500000 5.100000 11.200000 3.000000

```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

The `FindMaxAbsValue()` function performs the same computation but avoids modifying the `inputVals` array. The `FindMaxAbsValue()` also defines the `inputVals` as `const`, which prevents the function from modifying the parameter. For arrays that should not be modified in a function, good practice is to define the array parameter as `const`.

Figure 6.12.2: `FindMaxAbsValue()` computes the largest absolute value without modifying the array.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
#include <stdio.h>
#include <math.h>

float FindMaxAbsValue(const float inputVals[], int numVals) {
    int i;
    float maxAbsVal;
    float inputAbsVal;

    maxAbsVal = fabs(inputVals[0]);
    for (i = 0; i < numVals; ++i) {
        inputAbsVal = fabs(inputVals[i]);
        if (inputAbsVal > maxAbsVal) {
            maxAbsVal = inputAbsVal;
        }
    }
    return maxAbsVal;
}

int main(void) {
    const int NUM_VALUES = 5;
    // Array of changes in temperatures
    float tempChanges[5] = {10.0, 0.5, -5.1, -11.2, 3.0};
    float maxAbsChange;
    int i;

    // Print array before function call
    printf("tempChanges array before function call: ");
    for(i = 0; i < NUM_VALUES; ++i) {
        printf("%f ", tempChanges[i]);
    }
    printf("\n");

    // Find the largest temperature change, and print result.
    maxAbsChange = FindMaxAbsValue(tempChanges, NUM_VALUES);
    printf("Max absolute temperature change: %f\n", maxAbsChange);

    // Print array after function call
    printf("tempChanges array after function call: ");
    for(i = 0; i < NUM_VALUES; ++i) {
        printf("%f ", tempChanges[i]);
    }
    printf("\n");

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

tempChanges array before function call: 10.000000 0.500000 -5.100000 -11.200000 3.000000
Max absolute temperature change: 11.200000
tempChanges array after function call: 10.000000 0.500000 -5.100000 -11.200000 3.000000

PARTICIPATION ACTIVITY

6.12.3: Functions array parameters.



Based on the function names, should the following functions allow the array parameters to be modified?

1) `void PrintReverse(int inputVals[], int numVals)`

- Yes
- No

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



2) `void SortArrayAscending(int inputVals[], int numVals)`

- Yes
- No



3) `int FindIndexSmallest(int inputVals[], int numVals)`

- Yes
- No



6.13 Scope of variable/function definitions

The name of a defined variable or function item is only visible to part of a program, known as the item's **scope**. A variable declared in a function has scope limited to inside that function. In fact, because a compiler scans a program line-by-line from top-to-bottom, the scope starts *after* the declaration until the function's end. The following highlights the scope of local variable `cmVal`.

Figure 6.13.1: Local variable scope.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
#include <stdio.h>

const double CM_PER_IN = 2.54;
const int IN_PER_FT = 12;

/* Converts a height in feet/inches to centimeters */
double HeightFtInToCm(int heightFt, int heightIn) {
    int totIn;
    double cmVal;

    totIn = (heightFt * IN_PER_FT) + heightIn; // Total inches
    cmVal = totIn * CM_PER_IN; // Conv inch to cm
    return cmVal;
}

int main(void) {
    int userFt; // User defined feet
    int userIn; // User defined inches

    // Prompt user for feet/inches
    printf("Enter feet: ");
    scanf("%d", &userFt);

    printf("Enter inches: ");
    scanf("%d", &userIn);

    // Output the conversion result
    printf("Centimeters: %lf\n",
        HeightFtInToCm(userFt, userIn));

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490
 Jacob Adams
 UACS100Fall2021

Note that variable cmVal is invisible to the function main(). A statement in main() like newLen = cmVal; would yield a compiler error, e.g., the "error: cmVal was not declared in this scope". Likewise, variables userFt and userIn are invisible to the function HeightFtInToCm(). Thus, a programmer is free to define items with names userFt or userIn in function HeightFtInToCm().

A variable declared outside any function is called a **global variable**, in contrast to a *local variable* declared inside a function. A global variable's scope extends after the declaration to the file's end, and reaches into functions. For example, HeightFtInToCm() above accesses global variables CM_PER_IN and IN_PER_FT.

Global variables should be used sparingly. If a function's local variable (including a parameter) has the same name as a global variable, then in that function the name refers to the local item and the global is inaccessible. Such naming can confuse a reader. Furthermore, if a function updates a global variable, the function has effects that go beyond its parameters and return value, known as **side effects**, which make program maintenance hard. Global variables are typically limited to const variables like the number of centimeters per inch above. Beginning programmers sometimes use globals to avoid having to use parameters, which is bad practice. Good practice is to minimize the use of non-const global variables.

PARTICIPATION ACTIVITY

6.13.1: Variable/function scope.



- 1) A local variable is declared inside a function, while a global is declared outside any function.
- True
- False
- 2) A local variable's scope extends from a function's opening brace to the function's closing brace.
- True
- False
- 3) If a programmer declares a function's local variable to have the same name as a function parameter, the name will refer to the local variable.
- True
- False
- 4) If a programmer declares a function's local variable to have the same name as a global variable, the name will refer to the local variable.
- True
- False
- 5) A function that changes the value of a global variable is sometimes said to have "side effects".
- True
- False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

A function also has scope, which extends from its definition to the end of the file. Commonly, a programmer wishes to have the main() definition appear near the top of a file, with other functions definitions appearing further below, so that the main function is the first thing a reader sees. However, given function scope, main() would not be able to call any of those other functions. A solution involves function declarations. A **function declaration** specifies the function's return type, name, and parameters, ending with a semicolon where the opening brace would have gone. A function

declaration is also known as a **function prototype**. The function declaration gives the compiler enough information to recognize valid calls to the function. So by placing function declarations at the top of a file, the main function can then appear next, with actual function definitions appearing later in the file.

Figure 6.13.2: A function declaration allows a function definition to appear later in a file.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
#include <stdio.h>
#include <math.h> // To use "pow" function

/* Program to convert given-year U.S. dollars to
current dollars, using simplistic method of 4% annual inflation.
Source: http://inflationdata.com (See: Historical) */

// (Function DECLARATION)
double ToCurrDollars (double pastDol, int pastYr, int currYr);

int main(void) {
    double pastDol; // Starting dollar amount
    double currDol; // Ending dollar amount (converted value)
    int pastYr; // Starting year
    int currYr; // Ending year (converted to year)

    // Prompt user for previous year/dollar and current year
    printf("Enter current year: ");
    scanf("%d", &currYr);
    printf("Enter past year: ");
    scanf("%d", &pastYr);
    printf("Enter past dollars (Ex: 1000): ");
    scanf("%lf", &pastDol);

    // Function call to convert past to current dollars
    currDol = ToCurrDollars(pastDol, pastYr, currYr);

    printf("$%lf in %d is about $%lf in %d\n",
        pastDol, pastYr, currDol, currYr);

    return 0;
}

// (Function DEFINITION)
// Function returns equivalent value of pastDol in pastYr to currYr
double ToCurrDollars (double pastDol, int pastYr, int currYr) {
    double currDol; // Equivalent dollar amount given inflation

    currDol = pastDol * pow(1.04, currYr - pastYr );

    return currDol;
}
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
Enter current year: 2015
Enter past year: 1970
Enter past dollars (Ex: 1000): 10000
$10000 in 1970 is about $58411.756815 in 2015
(average annual U.S. income in 1970)
```

...

```
Enter current year: 2015
Enter past year: 1970
Enter past dollars (Ex: 1000): 23000
$23000 in 1970 is about $134347.040674 in 2015
(average U.S. house price in 1970)
```

...

```
Enter current year: 2015
Enter past year: 1933
Enter past dollars (Ex: 1000): 37
$37 in 1933 is about $922.434519 in 2015
(cost of Golden Gate Bridge, in millions)
```

...

```
Enter current year: 2015
Enter past year: 1969
Enter past dollars (Ex: 1000): 25
$25 in 1969 is about $151.870568 in 2015
(cost of Apollo space program, in billions)
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

A common error is for the function definition to not match the function declaration, such as a parameter defined as double in the declaration but as int in the definition, or with a slightly different identifier. The compiler detects such errors.

PARTICIPATION ACTIVITY

6.13.2: Function declaration and definition.



- 1) A function declaration lists the contents of a function, while a function definition just specifies the function's interface.

- True
 False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



- 2) A function declaration enables calls to the function before the function definition.

- True



False

Exploring further:

- [More on Scope](#) from msdn.microsoft.com

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

6.14 Parameter error checking

Verifying parameter values

Commonly, a function expects parameter values to be within some range. A good practice is to check that a parameter's value is within an expected range. If not in the range, the function might take one or more of various actions, like outputting an error message, assigning a valid value, returning a value indicating failure, exiting the program, etc.

Figure 6.14.1: Function with parameter error checking.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
#include <stdio.h>

void PrintDate(int currDay, int currMonth, int currYear) {

    // Parameter error checking
    if ((currDay < 1) || (currDay > 31)) {
        printf("Invalid day (%d). Using 1.\n", currDay);
        currDay = 1;
    }

    if ((currMonth < 1) || (currMonth > 12)) {
        printf("Invalid month (%d). Using 1.\n", currMonth);
        currMonth = 1;
    }

    // Begin function's normal behavior
    printf("%d/%d/%d", currMonth, currDay, currYear);
}

int main(void) {

    PrintDate(30, 7, 2012);
    printf("\n\n");

    PrintDate(40, 7, 2012);
    printf("\n\n");

    PrintDate(30, 13, 2012);
    printf("\n\n");

    return 0;
}
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

7/30/2012

Invalid day (40). Using 1.
7/1/2012Invalid month (13). Using 1.
1/30/2012**PARTICIPATION ACTIVITY**

6.14.1: Checking parameter values.

Consider the example above.

- 1) The month must be in the range from 1 to ____ .

Check**Show answer**

- 2) If the month parameter is not in the valid range, the code outputs an error message, and assigns currMonth with ____ .

Check**Show answer**©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



- 3) In addition to currMonth, which other parameter is checked for being in a valid range?

Check**Show answer**

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

6.15 Preprocessor and include

The **preprocessor** is a tool that scans the file from top to bottom looking for any lines that begin with #, known as a **hash symbol**. Each such line is not a program statement, but rather directs the preprocessor to modify the file in some way before compilation continues, each such line being known as a **preprocessor directive**. The directive ends at the end of the line, no semicolon is used at the end of the line.

Perhaps the most commonly-used preprocessor directive is **#include**, known as an **include directive**. #include directs the compiler to replace that line by the contents of the given filename.

Construct 6.15.1: Include directives.

```
#include "filename"  
#include <filename>
```

The following animation illustrates.

PARTICIPATION
ACTIVITY

6.15.1: Preprocessor's handling of an include directive.



Animation captions:

©zyBooks 12/22/21 12:35 1026490

1. The preprocessor replaces include by contents of myfile.h during compilation.

UACS100Fall2021

Good practice is to use a .h suffix for any file that will be included in another file. The h is short for header, to indicate that the file is intended to be included at the top (or header) of other files. Although any file can be included in any other file, convention is to only include .h files.

The characters surrounding the filename determine where the preprocessor looks for the file.

- `#include "myfile.h"` -- A filename in quotes causes the preprocessor to look for the file in the same folder/directory as the including file.
- `#include <stdfile.h>` -- A filename in angle brackets causes the preprocessor to look in the system's standard library folder/directory. Programmers typically use angle brackets only for standard library files, using quotes for all other include files. Note that nearly every previous example has included at least one standard library file, using angle brackets.

PARTICIPATION ACTIVITY**6.15.2: Include directives.**

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

- 1) The preprocessor processes any line beginning with what symbol?
 #
 <filename>
 "filename"
- 2) After a source file is processed by the preprocessor, is it correct to say that all hash symbols will be removed from the code remaining to be compiled?
 yes
 no



- 3) Do header files have to end in .h?
 yes
 no



- 4) Where does the preprocessor look for myfile.h in the line:

`#include "myfile.h"`

- Current folder
- System folder
- Unknown

- 5) What one symbol is incorrect in the following:

`#include <stdlib.h>;`

- #
- <>
- ;

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Exploring further:

- Preprocessor tutorial on cplusplus.com
- Preprocessor directives on MSDN

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

6.16 Separate files

Separating part of a program's code into a separate file can yield several benefits. One benefit is preventing a main file from becoming unmanageably large. Another benefit is that the separated part could be useful in other programs.

Suppose a program has several related functions that operate on triples of numbers, such as computing the maximum of three numbers or computing the average of three numbers. Those related functions' definitions can be placed in their own file as shown below in the file threeintsfcts.c.

Figure 6.16.1: Putting related functions in their own file.

main.c	threeintsfcts.c	
<pre>#include <stdio.h> #include "threeintsfcts.h" // Normally lots of other code here int main(void) { printf("%d\n", ThreeIntsSum(5, 10, 20)); printf("%d\n", ThreeIntsAvg(5, 10, 20)); return 0; } // Normally lots of other code here</pre>	<pre>int ThreeIntsSum(int num1, int num2, int num3) { return (num1 + num2 + num3); } int ThreeIntsAvg(int num1, int num2, int num3) { int sum; sum = num1 + num2 + num3; return (sum / 3); }</pre>	<pre>> a.out 35 11 ></pre>
	threeintsfcts.h	<pre>int ThreeIntsSum(int num1, int num2, int num3); int ThreeIntsAvg(int num1, int num2, int num3);</pre>

One could then compile the main.c and threeintsfcts.c files together as shown below.

Figure 6.16.2: Compiling multiple files together.

Without <code>#include "threeintsfcts.h"</code> in main.c	With <code>#include "threeintsfcts.h"</code> in main.c
<pre>> gcc -Wall main.c threeintsfcts.c main.c: In function 'main': main.c:8: warning: implicit declaration of function 'ThreeIntsSum' main.c:9: warning: implicit declaration of function 'ThreeIntsAvg'</pre>	<pre>> gcc -Wall main.c threeintsfcts.c ></pre>

Just compiling those two files (without the `#include "threeintsfcts.h"` line in the main file) would yield an error, as shown above on the left. The problem is that the compiler does not see the function definitions while processing the main file because those definitions are in another file, which is similar to what occurs when defining functions after `main()`. The solution for both situations is to provide function declarations before `main()` so the compiler knows enough about the functions to compile calls to those functions. Instead of typing the declarations directly above `main()`, a programmer can provide the function declarations in a header file, such as the `threeintsfcts.h` file provided in the figure above. The programmer then includes the contents of that file into a source file via the line: `#include "threeintsfcts.h"`.

The reader may note that the .h file could have contained function definitions rather than just function declarations, eliminating the need for two files (one for declarations, one for definitions). However, the two file approach has two key advantages. One advantage is that with the two file approach, the .h file serves as a brief summary of all functions available. A second advantage is that the main file's copy does not become exceedingly large during compilation, which can lead to slow compilation.

One last consideration that must be dealt with is that a header file could get included multiple times, causing the compiler to generate errors indicating an item defined in that header file is defined multiple times (the above header files only declared functions and didn't define them, but other header files may define functions, types, constants, and other items). Multiple inclusion commonly can occur when one header file includes another header file, e.g., the main file includes file1.h and file2.h, and file1.h also includes file2.h -- thus, file2.h would get included twice into the main file.

The solution is to add some additional preprocessor directives, known as header file guards, to the .h file as follows.

Construct 6.16.1: Header file guards.

```
#ifndef FILENAME_H
#define FILENAME_H

// Header file contents

#endif
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Header file guards are preprocessor directives, which cause the compiler to only include the contents of the header file once. `#define FILENAME_H` defines the symbol FILENAME_H to the preprocessor. The `#ifndef FILENAME_H` and `#endif` form a pair that instructs the preprocessor to process the code between the pair only if FILENAME_H is not defined ("ifndef" is short for "if not defined"). Thus, if the preprocessor includes encounter the header more than once, the code in the file during the second and any subsequent encounters will be skipped because FILENAME_H was already defined.

Good practice is to guard every header file. The following shows the threeintsfcts.h file with the guarding code added.

Figure 6.16.3: All header files should be guarded.

```
#ifndef THREEINTSFCTS_H
#define THREEINTSFCTS_H

int ThreeIntsSum(int num1, int num2, int num3);
int ThreeIntsAvg(int num1, int num2, int num3);

#endif
```

PARTICIPATION
ACTIVITY

6.16.1: Header files.



1) Header files must end with .h.

- True
- False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



2) Header files should contain function definitions for functions declared in another file.

- True
-



False

- 3) Guarding a header file prevents multiple inclusion of that file by the preprocessor.

True
 False

- 4) Is the following the correct two-line sequence to guard a file named myfile.h?

```
#ifdef MYFILE_H  
#define MYFILE_H
```

True
 False

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Exploring further:

- Preprocessor tutorial on cplusplus.com
- Preprocessor directives on MSDN

6.17 C example: Salary calculation with functions

zyDE 6.17.1: Calculate salary: Using functions.

Separating calculations into functions simplifies modifying and expanding programs.

The following program calculates the tax rate and tax to pay, using functions. One function returns a tax rate based on an annual salary.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

1. Run the program below with annual salaries of 40000, 60000, and 0.
2. Change the program to use a function to input the annual salary.
3. Run the program again with the same annual salaries as above. Are results the same?

[Load default template](#)

```

1 #include <stdio.h>
2 #include <stdbool.h>
3
4 double GetCorrespondingTableValue(int search, int baseTable[], double valueTable[])
5     double value;
6     int i;
7     bool keepLooking;
8
9     i = 0;
10    keepLooking = true;
11
12    while ((i < tableSize) && keepLooking) {
13        if (search <= baseTable[i]) {
14            value = valueTable[i];
15            keepLooking = false;
16        }
17        else {
18            ++i;

```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

40000 60000 0

Run

A solution to the above problem follows. The program was altered slightly to allow a zero annual salary and to end when a user enters a negative number for an annual salary.

zyDE 6.17.2: Calculate salary: Using functions (solution).

[Load default template](#)

```

1 #include <stdio.h>
2 #include <stdbool.h>
3
4 // Function to prompt for and input an integer
5 int PromptForInteger(const char userPrompt[])
6     int inputValue;
7
8     printf("%s: \n", userPrompt);
9     scanf("%d", &inputValue);
10
11    return inputValue;
12 }
13
14 // ****
15
16 // Function to get a value from one table based on a range in the other table

```

```
17 double GetCorrespondingTableValue(int search, int baseTable[], double val  
18     double value;
```

```
60000 40000 1000000  
-1
```

Run

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021



6.18 C example: Domain name validation with functions

zyDE 6.18.1: Validate domain names with functions.

Functions facilitate breaking down a large problem into a collection of smaller ones.

A **top-level domain** (TLD) name is the last part of an Internet domain name like .com in example.com. A **core generic top-level domain** (core gTLD) is a TLD that is either .com, .org, or .info. A **restricted top-level domain** is a TLD that is either .biz, .name, or .pro. A **second-level domain** is a single name that precedes a TLD as in apple in apple.com

The following program repeatedly prompts for a domain name and indicates whether the domain name is valid and has a core gTLD. For this program, a valid domain name has second-level domain followed by a TLD, and the second-level domain has these three characteristics:

1. Is 1-63 characters in length.
2. Contains only uppercase and lowercase letters or a dash.
3. Does not begin or end with a dash.

For this program, a valid domain name must contain only one period, such as apple.cor not support.apple.com. The program ends when the user presses just the Enter key in response to a prompt.

1. Run the program. Note that a restricted gTLD is not recognized as such.

2. Change the program by writing an input function and adding the validation for a restricted gTLD. Run the program again.

Load default template

```
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4 #include <stdbool.h>
5
6 // Global variables used for array Lengths
7 const int MAX_NUMS = 4;
8 const int MAX_SIZE = 6;
9
10 // ****
11
12 // Returns the position of a single period in a string
13 int GetPeriodPosition(char stringToSearch[]) {
14     int stringLength;
15     int periodCounter;
16     int periodPosition;
17     int i;
18 }
```

apple.com
APPLE.com
apple.comm
.

Run

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

zyDE 6.18.2: Validate domain names with functions.

A solution to the above problem follows.

Load default template

```
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4 #include <stdbool.h>
5 #include <stdlib.h>
6
7 // Global variables used for array Lengths
8 const int MAX_NUMS = 4;
9 const int MAX_SIZE = 6;
10
11 // ****
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

```
12  
13 // Returns the position of a single period in a string  
14 int GetPeriodPosition(char stringToSearch[]) {  
15     int stringLength;  
16     int periodCounter;  
17     int periodPosition;  
18     int i;
```

apple.com
APPLE.com
apple.comm

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Run

6.19 LAB: Flip a coin

Write a program that simulates flipping a coin to make decisions. The input is how many decisions are needed, and the output is either heads or tails. Assume the input is a value greater than 0.

Ex: If the input is:

3

the output is:

heads
tails
heads

For reproducibility needed for auto-grading, seed the program with a value of 2. In a real program, you would seed with the current time. In that case, every program's output would be different, which is what is desired but can't be auto-graded.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Note: A common student mistake is to call srand() before each call to rand(). But seeding should only be done once, at the start of the program, after which rand() can be called any number of times.

Your program must define and call the following function that randomly picks 0 or 1 and assigns decisionString with "heads" or "tails". Assume the value 0 represents "heads" and 1 represents "tails".

```
void HeadsOrTails(char* decisionString)
```

NaN.2052980.qx3zqy7



main.c

Load default template...

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 /* Define your function here */
6
7 int main(void) {
8     /* Add more variables as needed */
9
10    srand(2); /* Unique seed */
11
12    /* Type your code here. */
13
14    return 0;
15 }
16
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

Coding trail of your work

[What is this?](#)

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

History of your effort will appear here once you begin working on this zyLab.

6.20 LAB: Acronyms

An acronym is a word formed from the initial letters of words in a set phrase. Write a program whose input is a phrase and whose output is an acronym of the input. If a word begins with a lower case letter, don't include that letter in the acronym. Assume there will be at least one upper case letter in the input.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Ex: If the input is:

Institute of Electrical and Electronics Engineers

the output is:

IEEE

Your program must define and call a function that assigns userAcronym with the acronym created for the given userPhrase.

```
void CreateAcronym(char userPhrase[], char userAcronym[])
```

Hint: Refer to the ascii table to make sure a letter is upper case.

NaN.2052980.qx3zqy7

LAB ACTIVITY | 6.20.1: LAB: Acronyms 0 / 10

main.c Load default template...

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 /* Define your function here */
6
7 int main(void) {
8
9     /* Type your code here. */
10
11     return 0;
12 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

©zyBooks 12/22/21 12:35 10249 Output
(Your program)
Jacob Adams
UACS100Fall2021

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.21 LAB: Exact change - functions

Define a function called ExactChange that takes the total change amount in cents and an integer array as parameters. Function ExactChange() calculates the change using the fewest coins and stores the number of each coin type used into the array parameter. Index 0-3 of the array parameter should contain the number of pennies, nickels, dimes, and quarters respectively. Then write a main program that reads the total change amount as an integer input, calls ExactChange(), and outputs the change, one coin type per line. Use singular and plural coin names as appropriate, like 1 penny vs. 2 pennies. Output "no change" if the input is 0 or less.

Ex: If the input is:

0

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

(or less), the output is:

no change

Ex: If the input is:

45

the output is:

```
2 dimes
1 quarter
```

Your program must define and call the following function. Positions 0-3 of coinVals should contain the number of pennies, nickels, dimes, and quarters, respectively.

```
void ExactChange(int userTotal, int coinVals[])
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

NaN.2052980.qx3zqy7

LAB ACTIVITY

6.21.1: LAB: Exact change - functions

0 / 10



main.c

[Load default template...](#)

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. Your code must call the function. */
8
9     return 0;
10}
11|
```

[Develop mode](#)

[Submit mode](#)

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

[Run program](#)

Input (from above)



main.c
(Your program)



Outp

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

◀ ▶ ©zyBooks 12/22/21 12:35 1026490

◀ ▶ Jacob Adams
UACS100Fall2021

6.22 LAB: Even/odd values in an array

Write a program that reads a list of integers, and outputs whether the list contains all even numbers, odd numbers, or neither. The input begins with an integer indicating the number of integers in the list. The first integer is not in the list. Assume that the list will always contain less than 20 integers.

Ex: If the input is:

5 2 4 6 8 10

the output is:

all even

Ex: If the input is:

5 1 -3 5 -7 9

the output is:

all odd

Ex: If the input is:

5 1 2 3 4 5

the output is:

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

not even or odd

Your program must define and call the following two functions. IsArrayEven returns true if all integers in the array are even and false otherwise. IsArrayOdd returns true if all integers in the array are odd and false otherwise.

```
bool IsArrayEven(int inputVals[], int numVals)
bool IsArrayOdd(int inputVals[], int numVals)
```

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.22.1: LAB: Even/odd values in an array

0 / 10



©zyBooks 12/22/21 12:35 1026490

Jacob Adams

Load default template...

UACS100Fall2021

main.c

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 /* Define your function here */
5
6 int main(void) {
7
8     /* Type your code here. Remember to include the bool Library*/
9
10    return 0;
11 }
12
```

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

main.c
(Your program)

Output

Program output displayed here

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.23 LAB: Miles to track laps



This section has been set as optional by your instructor.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

One lap around a standard high-school running track is exactly 0.25 miles. Write a program that takes a number of miles as input, and outputs the number of laps.

Output each floating-point value with two digits after the decimal point, which can be achieved as follows:

```
printf("%0.2lf", yourValue);
```

Ex: If the input is:

1.5

the output is:

6.00

Ex: If the input is:

2.2

the output is:

8.80

Your program must define and call a function:

```
double MilesToLaps(double userMiles)
```

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.23.1: LAB: Miles to track laps

©zyBooks 12/22/21 12:35 1026490
Jacob Adams 0 / 10
UACS100Fall2021

main.c

Load default template...

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
```

```
5 int main(void) {  
6  
7     /* Type your code here. Your code must call the function. */  
8  
9     return 0;  
10 }  
11 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)



main.c
(Your program)

→ Outp

Program output displayed here

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.24 LAB: Step counter

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



This section has been set as optional by your instructor.

A pedometer treats walking 1 step as walking 2.5 feet. Define a function named FeetToSteps that takes a double as a parameter, representing the number of feet walked, and returns an integer that

represents the number of steps walked. Then, write a main program that reads the number of feet walked as an input, calls function FeetToSteps() with the input as an argument, and outputs the number of steps.

Use floating-point arithmetic to perform the conversion.

Ex: If the input is:

150.5

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

the output is:

60

The program must define and call a function:

```
int FeetToSteps(double userFeet)
```

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.24.1: LAB: Step counter

0 / 10



main.c

Load default template...

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. */
8
9     return 0;
10 }
11
```

©zyBooks 12/22/21 12:35 1026490

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

Coding trail of your work

[What is this?](#)

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

History of your effort will appear here once you begin working on this zyLab.



6.25 LAB: A jiffy



This section has been set as optional by your instructor.

A "jiffy" is the scientific name for 1/100th of a second. Given an input number of seconds, output the number of "jiffies."

Output each floating-point value with two digits after the decimal point, which can be achieved as follows:

```
printf("%0.2lf", yourValue);
```

Ex: If the input is:

15

the output is:

1500.00

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Your program must define and call a function:

```
double SecondsToJiffies(double userSeconds)
```

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.25.1: LAB: A jiffy

0 / 10



main.c

[Load default template...](#)

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. */
8
9     return 0;
10 }
11
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

[Develop mode](#)[Submit mode](#)

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

[Run program](#)

Input (from above)



main.c
(Your program)



Outp

Program output displayed here

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

6.26 LAB: Driving cost - functions



This section has been set as optional by your instructor.

Write a function DrivingCost with input parameters drivenMiles, milesPerGallon, and dollarsPerGallon, that returns the dollar cost to drive those miles. All items are of type double. If the function is called with 50 20.0 3.1599, the function returns 7.89975.

Define that function in a program whose inputs are the car's miles/gallon and the gas dollars/gallon (both doubles). Output the gas cost for 10 miles, 50 miles, and 400 miles, by calling your DrivingCost function three times.

Output each floating-point value with two digits after the decimal point, which can be achieved as follows:

```
printf("%0.2f", yourValue);
```

Ex: If the input is:

```
20.0 3.1599
```

the output is:

```
1.58 7.90 63.20
```

Your program must define and call a function:

```
double DrivingCost(double drivenMiles, double milesPerGallon, double  
dollarsPerGallon)
```

NaN.2052980.qx3zqy7

LAB ACTIVITY

6.26.1: LAB: Driving cost - functions

0 / 10



main.c

Load default template...

```
1 #include <stdio.h>  
2 /* Define your function here */  
3  
4 int main(void) {  
5     /* Type your code here. */  
6  
7     return 0;  
8 }  
9  
10  
11
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed hereCoding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.27 LAB: Swapping variables



This section has been set as optional by your instructor.

Write a program whose input is two integers and whose output is the two integers swapped.

Ex: If the input is:

3 8

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

then the output is:

8 3

Your program must define and call a function. SwapValues returns the two values in swapped order.

```
void SwapValues(int* userVal1, int* userVal2)
```

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.27.1: LAB: Swapping variables

0 / 10

**main.c**[Load default template...](#)

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. Your code must call the function. */
8
9     return 0;
10 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.28 LAB: Max magnitude



This section has been set as optional by your instructor.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACSTOOFall2021

Write a function MaxMagnitude with two integer input parameters that returns the largest magnitude value. Use the function in a program that takes two integer inputs, and outputs the largest magnitude value.

Ex: If the inputs are:

5 7

the function returns:

7

Ex: If the inputs are:

-8 -2

the function returns:

-8

Note: The function does not just return the largest value, which for -8 -2 would be -2. Though not necessary, you may use the absolute-value built-in math function.

Your program must define and call a function:

`int MaxMagnitude(int userVal1, int userVal2)`

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.28.1: LAB: Max magnitude

0 / 10

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACSTOOFall2021
[Load default template...](#)

main.c

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. */
8 }
```

```
9     return 0;  
10 }  
11 |
```

©zyBooks 12/22/21 12:35 1026490 ↗

Jacob Adams
UACS100Fall2021**Develop mode****Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

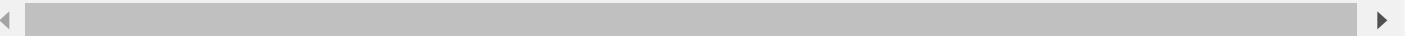
Input (from above)

**main.c**
(Your program)

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.29 LAB: Count characters - functions

©zyBooks 12/22/21 12:35 1026490 ↗

Jacob Adams
UACS100Fall2021

This section has been set as optional by your instructor.

Write a program whose input is a character and a string, and whose output indicates the number of times the character appears in the string. You may assume that the string does not contain spaces and will always contain less than 50 characters.

Ex: If the input is:

n Monday

the output is:

1

Ex: If the input is:

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

z today

the output is:

0

Ex: If the input is:

n Sunny

the output is:

2

Case matters. n is different than N.

Ex: If the input is:

n Nobody

the output is:

0

Your program must define and call the following function that returns the number of times the input character appears in the input string. **int CountCharacters(char userChar, char* userString)**

NaN.2052980.qx3zqy7

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021 0 / 10



LAB ACTIVITY

6.29.1: LAB: Count characters - functions

main.c

Load default template...

```
1 #include <stdio.h>
2 #include <string.h>
3
4 /* Define your function here */
```

```
5  
6 int main(void) {  
7  
8     /* Type your code here. */  
9  
10    return 0;  
11 }  
12
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)



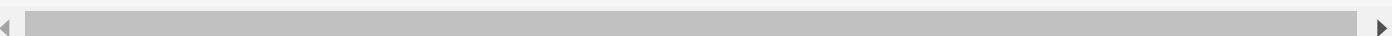
main.c
(Your program)

→ Output

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.30 LAB: Remove non-alphabetic characters - functions

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



This section has been set as optional by your instructor.

Write a program that removes all non-alphabetic characters from the given input. You may assume that the input string will not exceed 50 characters.

Ex: If the input is:

```
-Hello, 1 world$!
```

the output is:

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
Helloworld
```

Your program must define and call the following function. userString is the user specified string. The function assigns userStringAlphaOnly with the user specified string without non-alphabetic characters.

```
void RemoveNonAlpha(char userString[], char userStringAlphaOnly[])
```

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.30.1: LAB: Remove non-alphabetic characters - functions

0 / 10



main.c

[Load default template...](#)

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 /* Define your function here */
6
7 int main(void) {
8
9     /* Type your code here. */
10
11    return 0;
12 }
13
```

©zyBooks 12/22/21 12:35 1026490

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

Coding trail of your work

[What is this?](#)

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

History of your effort will appear here once you begin working on this zyLab.



6.31 LAB: Convert to binary - functions



This section has been set as optional by your instructor.

Write a program that takes in a positive integer as input, and outputs a string of 1's and 0's representing the integer in binary. For an integer x , the algorithm is:

```
As long as x is greater than 0
    Output x % 2 (remainder is either 0 or 1)
    x = x / 2
```

Note: The above algorithm outputs the 0's and 1's in reverse order. You will need to write a second function to reverse the string.

Ex: If the input is:

6

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

the output is:

110

Your program must define and call the following two functions. The `IntegerToReverseBinary` function should assign `binaryValue` with a string of 1's and 0's representing the `integerValue` in binary (in reverse order). The `ReverseString` function should assign `reversedString` with the reverse of `inputString`.

```
void IntegerToReverseBinary(int integerValue, char binaryValue[])
void ReverseString(char inputString[], char reversedString[])
```

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.31.1: LAB: Convert to binary - functions

0 / 10



©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

main.c

Load default template...

```
1 #include <stdio.h>
2
3 /* Define your functions here */
4
5 int main(void) {
6
7     /* Type your code here. Your code must call the functions. */
8
9     return 0;
10}
11
```

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.32 LAB: Max and min numbers



This section has been set as optional by your instructor.

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Write a program whose inputs are three integers, and whose outputs are the largest of the three values and the smallest of the three values.

Ex: If the input is:

```
7 15 3
```

the output is:

```
largest: 15
smallest: 3
```

Your program must define and call the following two functions. The LargestNumber function should return the largest number of the three input values. The SmallestNumber function should return the smallest number of the three input values.

```
int LargestNumber(int userNum1, int userNum2, int userNum3)
int SmallestNumber(int userNum1, int userNum2, int userNum3)
```

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.32.1: LAB: Max and min numbers

0 / 10



main.c

Load default template...

```
1 #include <stdio.h>
2 /* Define your function here */
3
4 int main(void) {
5     /* Type your code here. Your code must call the function. */
6
7     return 0;
8 }
9
10
11
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed hereCoding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.33 LAB: Leap year - functions



This section has been set as optional by your instructor.

A year in the modern Gregorian Calendar consists of 365 days. In reality, the earth takes longer to rotate around the sun. To account for the difference in time, every 4 years, a leap year takes place. A leap year is when a year has 366 days: An extra day, February 29th. The requirements for a given year to be a leap year are:

- 1) The year must be divisible by 4
- 2) If the year is a century year (1700, 1800, etc.), the year must be evenly divisible by 400

Some example leap years are 1600, 1712, and 2016.

Write a program that takes in a year and determines whether that year is a leap year.

Ex: If the input is:

```
1712
```

the output is:

```
1712 is a leap year.
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams
UACS100Fall2021

Ex: If the input is:

```
1913
```

the output is:

```
1913 is not a leap year.
```

Your program must define and call the following function. The function should return true if the input year is a leap year and false otherwise.

```
bool IsLeapYear (int userYear)
```

NaN.2052980.qx3zqy7

LAB ACTIVITY

6.33.1: LAB: Leap year - functions

0 / 10



main.c

Load default template...

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 /* Define your function here */
5
6 int main(void) {
7
8     /* Type your code here. */
9
10    return 0;
11 }
12
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

©zyBooks 12/22/21 12:35 10249 Output
(Your program)
Jacob Adams
UACS100Fall2021

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.34 LAB: Sort an array



This section has been set as optional by your instructor.

Write a program that gets a list of integers from input, and outputs the integers in ascending order (lowest to highest). The first integer indicates how many numbers are in the list. Assume that the list will always contain less than 20 integers.

Ex: If the input is:

5 10 4 39 12 2

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

the output is:

2 4 10 12 39

For coding simplicity, follow every output value by a space, including the last one.

Your program must define and call the following function. When the SortArray function is complete, the array passed in as the parameter should be sorted.

```
void SortArray(int sortingList[], int numVals)
```

Hint: There are many ways to sort an array. You are welcome to look up and use any existing algorithm. Some believe the simplest to code is bubble sort: https://en.wikipedia.org/wiki/Bubble_sort. But you are welcome to try others: https://en.wikipedia.org/wiki/Sorting_algorithm.

NaN.2052980.qx3zqy7

LAB ACTIVITY

6.34.1: LAB: Sort an array

©zyBooks 12/22/21 12:35 1026490
Jacob Adams 0 / 10
UACS100Fall2021

main.c[Load default template...](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     /* Type your code here. */
5
6     return 0;
7 }
8
9
```

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

main.c
(Your program)
©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Program output displayed here

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

◀ ▶

©zyBooks 12/22/21 12:35 1026490 ▶

Jacob Adams

UACS100Fall2021

6.35 LAB: Output values below an amount - functions



This section has been set as optional by your instructor.

Write a program that first gets a list of integers from input. The input begins with an integer indicating the number of integers that follow. Then, get the last value from the input, and output all integers less than or equal to that value. Assume that the list will always contain less than 20 integers.

Ex: If the input is:

5 50 60 140 200 75 100

the output is:

50 60 75

The 5 indicates that there are five integers in the list, namely 50, 60, 140, 200, and 75. The 100 indicates that the program should output all integers less than or equal to 100, so the program outputs 50, 60, and 75.

For coding simplicity, follow every output value by a space, including the last one.

Such functionality is common on sites like Amazon, where a user can filter results.

Write your code to define and use two functions. The GetUserValues function reads in numValues integers from the input and assigns to userValues. The OutputIntsLessThanOrEqualToThreshold function outputs all integers in userValues that are less than or equal to maxVal. numValues indicates the number of integers in userValues.

```
void GetUserValues(int userValues[], int numValues)
void OutputIntsLessThanOrEqualToThreshold(int userValues[], int maxVal, int
numValues)
```

Utilizing functions will help to make main() very clean and intuitive.

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.35.1: LAB: Output values below an amount - functions

0 / 10

**main.c**[Load default template...](#)

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. */
8
9     return 0;
10}
11
```

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Outp

Program output displayed here

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.36 LAB: Adjust list by normalizing - functions



This section has been set as optional by your instructor.

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

When analyzing data sets, such as data for human heights or for human weights, a common step is to adjust the data. This can be done by normalizing to values between 0 and 1, or throwing away outliers.

For this program, adjust the values by subtracting the smallest value from all the values. The input begins with an integer indicating the number of integers that follow. Assume that the list will always contain less than 20 integers.

Ex: If the input is:

5 30 50 10 70 65

the output is:

20 40 0 60 55

The 5 indicates that there are five values in the list, namely 30, 50, 10, 70, and 65. 10 is the smallest value in the list, so is subtracted from each value in the list.

For coding simplicity, follow every output value by a space, including the last one.

Your program must define and use the following function. The GetMinimumInt() function returns the smallest value in listInts. numVals indicates the number of values in listInts.

```
int GetMinimumInt(int listInts[], int numVals)
```

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.36.1: LAB: Adjust list by normalizing - functions

0 / 10



main.c

Load default template...

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

```
1 #include <stdio.h>
2
3 /* Define your function here */
4
5 int main(void) {
6
7     /* Type your code here. */
8
9     return 0;
10 }
```

11

#

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)

**main.c**
(Your program)

Output

Program output displayed here

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.

6.37 LAB: Warm up: Text analyzer & modifier



This section has been set as optional by your instructor.

©zyBooks 12/22/21 12:35 1026490Jacob AdamsUACS100Fall2021

- (1) Prompt the user to enter a string of their choosing, assuming a 50 character maximum. Output the string. (1 pt)

Ex:

Enter a sentence or phrase:

The only thing we have to fear is fear itself.

You entered: The only thing we have to fear is fear itself.

(2) Complete the GetNumOfCharacters() function, which returns the number of characters in the user's string. Make sure to count the trailing newline character if applicable. We encourage you to use a *for loop in this function*. (2 pts)

(3) In main(), call the GetNumOfCharacters() function and then output the returned result. (1 pt)

(4) Implement the OutputWithoutWhitespace() function. OutputWithoutWhitespace() outputs the string's characters except for whitespace (spaces, tabs). Note: A tab is '\t'. Call the OutputWithoutWhitespace() function in main(). (2 pts)

Ex:

Enter a sentence or phrase:

The only thing we have to fear is fear itself.

You entered: The only thing we have to fear is fear itself.

Number of characters: 47

String with no whitespace: Theonlythingwehavetofearisfearitself.

NaN.2052980.qx3zqy7

**LAB
ACTIVITY**

6.37.1: LAB: Warm up: Text analyzer & modifier

0 / 6

main.c

Load default template...

```
1 #include<stdio.h>
2 #include <string.h>
3
4 //Returns the number of characters in usrStr
5 int GetNumOfCharacters(const char usrStr[]) {
6
7     /* Type your code here. */
8
9 }
10
11 int main(void) {
12
13     /* Type your code here. */
14
15     return 0;
16 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

[Develop mode](#)[Submit mode](#)

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

[Run program](#)

Input (from above)

UACS100Fall2021
main.c
(Your program)

Outp

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.38 LAB*: Program: Authoring assistant



This section has been set as optional by your instructor.

- (1) Prompt the user to enter a string of their choosing. Store the text in a string. Output the string. (1 pt)

Ex:

Enter a sample text:

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

we'll continue our quest in space. there will be more shuttle flights and more shuttle crews and, yes, more volunteers, more civilians, more teachers in space. nothing ends here; our hopes and our journeys continue!

You entered: we'll continue our quest in space. there will be more shuttle flights and more shuttle crews and, yes, more volunteers,

more civilians, more teachers in space. nothing ends here; our hopes and our journeys continue!

(2) Implement the PrintMenu() function to print the following command menu. (1 pt)

Ex:

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

MENU

c - Number of non-whitespace characters
w - Number of words
f - Fix capitalization
r - Replace all !'s
s - Shorten spaces
q - Quit

(3) Implement the ExecuteMenu() function that takes 2 parameters: a character representing the user's choice and the user provided sample text. ExecuteMenu() performs the menu options, according to the user's choice, by calling the appropriate functions described below. (1 pt)

(4) In the main() function, call PrintMenu() and prompt for the user's choice of menu options for analyzing/editing the string. Each option is represented by a single character.

If an invalid character is entered, continue to prompt for a valid choice. When a valid option is entered, execute the option by calling ExecuteMenu(). Then, print the menu, and prompt for a new option. Continue until the user enters 'q'. *Hint: Implement Quit before implementing other options.* (1 pt)

Ex:

MENU

c - Number of non-whitespace characters
w - Number of words
f - Fix capitalization
r - Replace all !'s
s - Shorten spaces
q - Quit

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Choose an option:

(5) Implement the GetNumOfNonWSCharacters() function. GetNumOfNonWSCharacters() has a constant string as a parameter and returns the number of characters in the string, excluding all whitespace. *Hint: Using fgets() to read input will cause the input string to have a newline character at*

the end. The newline character should not be counted by `GetNumOfNonWSCharacters()`. Call `GetNumOfNonWSCharacters()` in the `ExecuteMenu()` function, and then output the returned value. (4 pts)

Ex:

Enter a sample text:

we'll continue our quest in space. there will be more shuttle flights
and more shuttle crews and, yes, more volunteers, more civilians,
more teachers in space. nothing ends here; our hopes and our journeys
continue!

You entered: we'll continue our quest in space. there will be more shuttle flights and more shuttle crews and, yes, more volunteers, more civilians, more teachers in space. nothing ends here; our hopes and our journeys continue!

MENU

c - Number of non-whitespace characters
w - Number of words
f - Fix capitalization
r - Replace all !'s
s - Shorten spaces
q - Quit

Choose an option:

c

Number of non-whitespace characters: 181

(6) Implement the `GetNumOfWords()` function. `GetNumOfWords()` has a constant string as a parameter and returns the number of words in the string. Hint: Words end when a space is reached except for the last word in a sentence. Call `GetNumOfWords()` in the `ExecuteMenu()` function, and then output the returned value. (3 pts)

Ex:

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Number of words: 35

(7) Implement the `FixCapitalization()` function. `FixCapitalization()` has a string parameter and updates the string by replacing lowercase letters at the beginning of sentences with uppercase letters. `FixCapitalization()` DOES NOT output the string. Call `FixCapitalization()` in the `ExecuteMenu()` function, and then output the edited string. (3 pts)

Ex:

Edited text: We'll continue our quest in space. There will be more shuttle flights and more shuttle crews and, yes, more volunteers, more civilians, more teachers in space. Nothing ends here; our hopes and our journeys continue!

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

(8) Implement the ReplaceExclamation() function. ReplaceExclamation() has a string parameter and updates the string by replacing each '!' character in the string with a '.' character. ReplaceExclamation() DOES NOT output the string. Call ReplaceExclamation() in the ExecuteMenu() function, and then output the edited string. (3 pts)

Ex.

Edited text: we'll continue our quest in space. there will be more shuttle flights and more shuttle crews and, yes, more volunteers, more civilians, more teachers in space. nothing ends here; our hopes and our journeys continue.

(9) Implement the ShortenSpace() function. ShortenSpace() has a string parameter and updates the string by replacing all sequences of 2 or more spaces with a single space. ShortenSpace() DOES NOT output the string. Call ShortenSpace() in the ExecuteMenu() function, and then output the edited string. (3 pt)

Ex:

Edited text: we'll continue our quest in space. there will be more shuttle flights and more shuttle crews and, yes, more volunteers, more civilians, more teachers in space. nothing ends here; our hopes and our journeys continue!

NaN.2052980.qx3zqy7

LAB
ACTIVITY

6.38.1: LAB*: Program: Authoring assistant

©zyBooks 12/22/21 12:35 1026490
Jacob Adams 0 / 20
UACS100Fall2021

main.c

Load default template...

```
1 #include<stdio.h>
2 #include <string.h>
3
4 /* Define your functions here. */
5
```

```
6 int main(void) {  
7     /* Type your code here. */  
8  
9     return 0;  
10 }  
11 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)



main.c
(Your program)



Output

Program output displayed here

Coding trail of your work

[What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



6.39 Lab training: Unit tests to evaluate your program

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021



This section has been set as optional by your instructor.

Auto-graded programming assignments may use a *Unit test* to test small parts of a program. Unlike a *Compare output test*, which evaluates your program's output for specific input values, a *Unit test* evaluates individual functions to determine if each function:

- is named correctly and has the correct parameters and return type
- calculates and returns the correct value (or prints the correct output)

This example lab uses multiple unit tests to test the KiloToPounds() function.

12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

Complete a program that takes a weight in kilograms as input, converts the weight to pounds, and then outputs the weight in pounds. 1 kilogram = 2.204 pounds (lbs).

Ex: If the input is:

10

the output is:

22.040000 lbs

Note: Your program must define the function

`double KiloToPounds(double kilos)`

The program below has an error in the KiloToPounds() function.

1. Try submitting the program for grading (click "Submit mode", then "Submit for grading"). Notice that the first two test cases fail, but the third test case passes. The first test case fails because the program outputs the result from the KiloToPounds() function, which has an error. The second test case uses a *Unit test* to test the KiloToPounds() function, which fails.
2. Change the KiloToPounds() function to multiply the variable kilos by 2.204, instead of dividing. The return statement should be: `return (kilos * 2.204);` Submit again. Now the test cases should all pass.

Note: A common error is to mistype a function name with the incorrect capitalization. Function names are case sensitive, so if a lab program asks for a KiloToPounds() function, a kiloToPounds() function that works for you in develop mode will result in a failed unit test (the unit test will not be able to find KiloToPounds()).

NaN.2052980.qx3zqy7

©zyBooks 12/22/21 12:35 1026490

Jacob Adams

UACS100Fall2021

LAB ACTIVITY

6.39.1: Lab training: Unit tests to evaluate your program

0 / 3



main.c

Load default template...

```
1 #include <stdio.h>
2
```

```
3 double KiloToPounds(double kilos) {  
4     // This statement intentionally has an error.  
5     return (kilos / 2.204);  
6 }  
7  
8 int main() {  
9     double kilos;  
10    double pounds;  
11  
12    scanf("%lf", &kilos);  
13  
14    pounds = KiloToPounds(kilos);  
15    printf("%f lbs\n", pounds);  
16  
17    return 0;  
18 }
```

©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021

Develop mode

Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above)



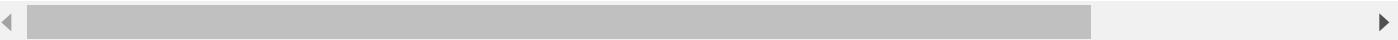
main.c
(Your program)

→ Outp

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.



©zyBooks 12/22/21 12:35 1026490
Jacob Adams
UACS100Fall2021